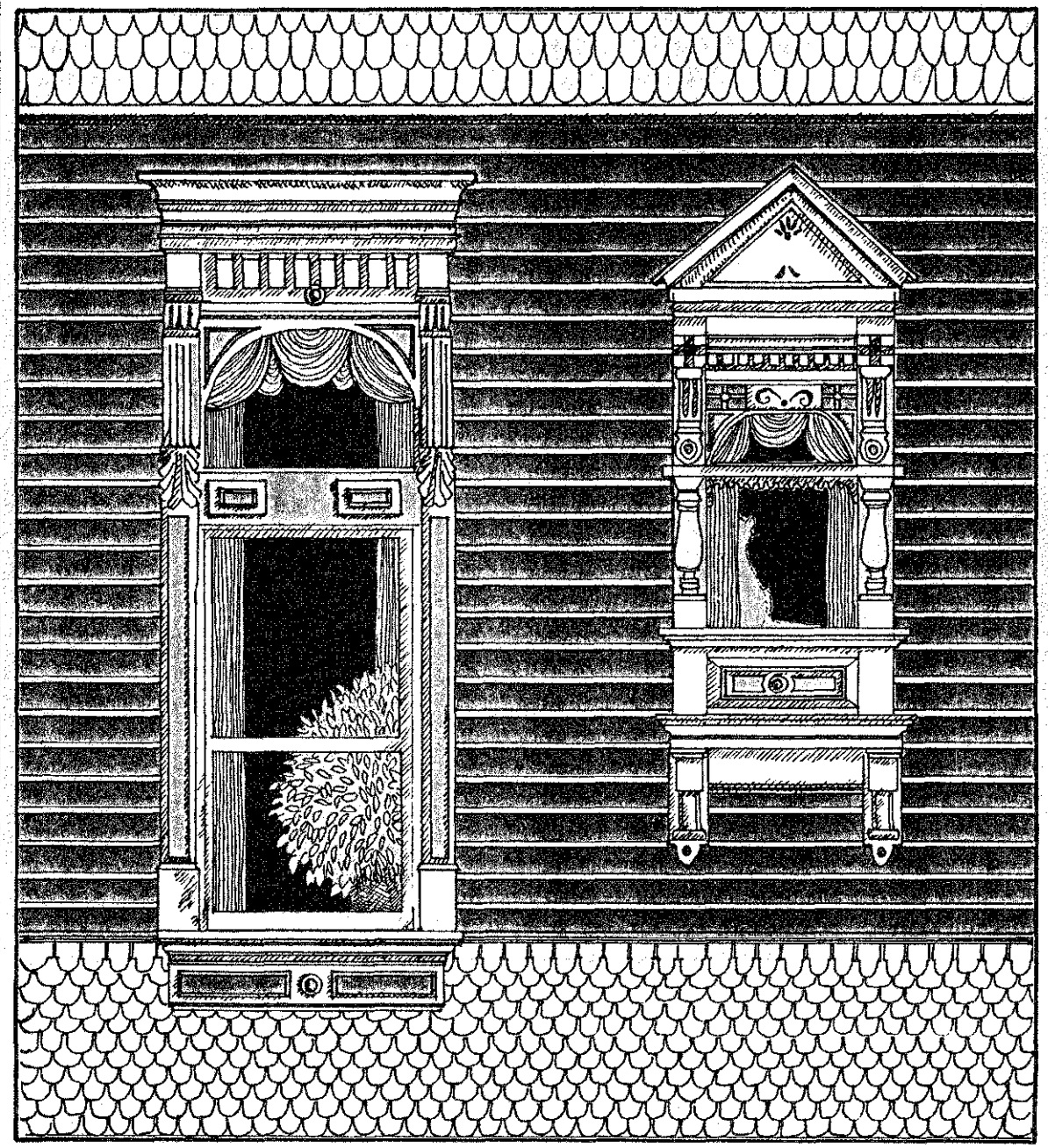
Vol. 4, No. 2 March/April 1986 $4.00

programmer's journal

The Resource Journal for IBM PC Programmers

THE HOME OFFICE SYNDROME

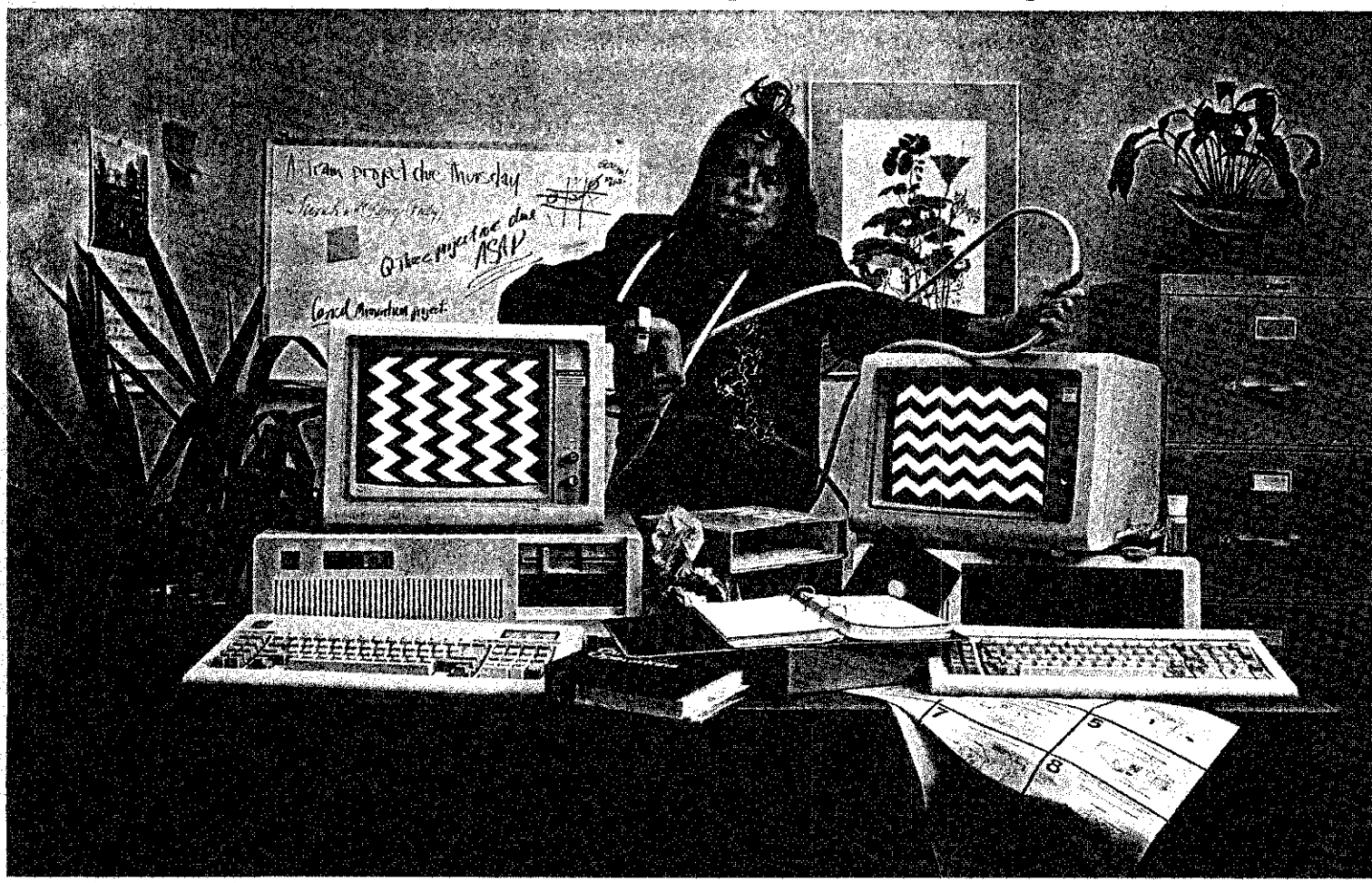


OFFICE COPY

BULK RATE
U.S. POSTAGE
PAID
EUGENE, OR 97401
PERMIT NO. 668

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

If you can't share files on your network, you're using the wrong file manager.



Be connected. Btrieve.®

Networks can solve problems. But running a single-user file manager can create new ones: Lost updates. Garbled data. Trashed files.

Btrieve® /N offers *safe* multi-user file management that protects your data when sharing files. And eliminates the need to rewrite your application for networking. Btrieve/N set the file management standard for the industry's most popular LANS: IBM's PC Network, Netware, PC Net and Ether-Series. And now Btrieve/N sets the standard for multi-user systems: XENIX, MultiLink Advanced and others.

Fast. Btrieve/N is fast, too. It's written in assembly language especially for the IBM PC. And based on b-tree file indexing with automatic balancing for access speed that won't degrade as your database grows. With Btrieve/N, your applications always run fast. And users don't waste time waiting.

Automatic file recovery. Btrieve/N provides automatic file recovery after a system crash, so accidents and power failures don't turn into database disasters. Your Btrieve data always comes back intact.

Fully-relational data management. SoftCraft's entire family of products gives you a complete, fully-relational database management system. Rtrieve™/N adds report writing capabilities for generating the reports you need. Xtrieve™/N speeds users through database queries with interactive, on-screen menus—no command language or special syntax.

For professional programmers. Btrieve/N is the fast, reliable answer for all your application development. In any development language—BASIC, Pascal, COBOL, C, FORTRAN and APL. With multikey access to records. Unlimited records per file. Duplicate, modifiable, and segmented keys.

With Btrieve/N, you can develop better network applications. And solve problems, not create new ones.

NO ROYALTIE

Suggested retail prices: Btrieve, $245; Btrieve/N, $595; Xtrieve, $195; Xtrieve/N, $395; Rtrieve, $85; Rtrieve/N, $175. Requires PC-DOS or MS-DOS 1.X, 2.X, or 3.X.

Btrieve is a registered trademark and Xtrieve a Rtrieve are trademarks of SoftCraft Inc.

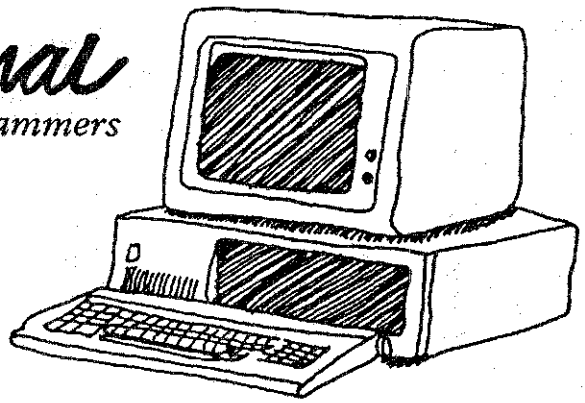
SoftCraft Inc.
P.O. Box 9802 #917
Austin, Texas 78766
(512) 346-8380 Telex 358 200

Circle Number **21** on the Reader Service C

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programmer's journal

The Resource Journal for IBM PC Programmers



Volume 4, Number 2 March/April 1986

CONTENTS

NOTE FROM THE EDITOR 2

LETTERS 4

PJ FORUM: THE INTEL 80386, SINGLE- VS. MULTI-USER, MULTI-TASKING COMPUTING, BENCHMARKS by FNE 5

Excerpts from DTACK GROUNDED: The Journal of Simple 68000/32081 Systems

CONSULTING: THE HOME OFFICE SYNDROME by Paul Barkley 12

How professional can you seem when you work out of your home?

REDIRECTIONS: INDUSTRY NEWS by Frank D. Greco 15

Wars and rumors of wars

PASCAL: FUNKY STRING FUNCTIONS by Tom Swan 22

Put some "soul" in your programming

PASCAL: CAPTURING FILE HANDLES IN TURBO PASCAL by Neil J. Rubenking 26

More file handling routines from Neil Rubenking

C: THE PROPOSED ANSI C STANDARD by Rex Jaeschke 30

The first of a PJ *series on the new ANSI Standard*

ASM: OPTIMIZING FOR SPEED by Michael Hoyt 32

Faster screen dumps to the printer

UTILITIES: MEMORY RESIDENT CONTROL OVER PRINT SCREEN FUNCTIONS by Steven Baker 38

More on the "dreaded locked keyboard" condition

BASIC: STRUCTURING BASIC OUTPUT WITH FIELD STATEMENTS by Bernard H. Robinson, Jr. 43

Position your ASCII data on the screen or printer

BASIC: NETWORKING—WHAT'S THE PROBLEM? by Robert R. Shaffer 48

A BASIC fan finds hidden worth in Microsoft's QuickBASIC Compiler

BOOKS: PROGRAMMING FOR PRINTER OUTPUT review by Michael Hoyt 50

The first volume of a proposed series

PJ BILLBOARD 52

HOUSEKEEPING 53

Author Guidelines, Advertiser Index, Peer Review

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programmer's journal
The Resource Journal for IBM PC Programmers

Publisher:	Elizabeth Oakley
Editor:	Robert Keller
Managing Editor:	Erin O'Connor
Production Manager:	Ralph Bentley
Circulation/Promotion:	Robert Tappan
Cover:	Julia King
Illustrations:	Julia King
Production Assistance:	Holly Tracy

Contributing Writers:

Steven Baker	Bernard H. Robinson, Jr.
Paul Barkley	Neil J. Rubenking
Frank D. Greco	Robert R. Shaffer
Michael Hoyt	Tom Swan
Rex Jaeschke	FNE

Founder: Greg Estes

Business and Editorial Offices:	Programmer's Journal 29 West 29th Ave., Suite 5 Eugene, OR 97405 or P.O. Box 30160 Eugene, OR 97403 (503) 484-2162
Subscription Inquiries:	Programmer's Journal P.O. Box 3000 Dept. EE Denville, NJ 07834 (503) 484-2162

Subscriptions: Bi-monthly rates $24 U.S.
($30 in Canada and $40 elsewhere)
2 years $44 U.S.
($55 in Canada and $75 elsewhere)
3 years $64 U.S.
($80 in Canada and $110 elsewhere)

Articles and letters in Programmer's Journal represent the views of their authors and are not necessarily those of the publisher. Letters to the editor become the property of Programmers Journal and may be published.

Advertising: For ad rates, deadlines, and placement, contact Ralph Bentley or Robin Tappan, (503) 484-2162, or write to Programmer's Journal, P.O. Box 30160, Eugene, OR 97403.

IBM and Personal Computer are trademarks of International Business Machines; dBASE II and dBASE III are trademarks of Ashton-Tate.

Programmer's Journal (ISSN 0747-5861) is published bi-monthly by Oakley Publishing Company. Application to mail at second-class rates is pending at Eugene, OR. POSTMASTER: Send address changes to Programmer's Journal, P.O. Box 3000, Dept EE., Denville, NJ 07834.
© 1986

NOTE FROM THE EDITOR

In This Issue

This issue's consulting article from Paul Barkley shows how your office space makes you look to a prospective client. Read about the trade-offs of a home office, a shared space, and various other arrangements.

Steve Baker has written a resident utility that controls the print screen keys. It's another cure for the dreaded locked keyboard condition.

Neil Rubenking shows how to obtain file handles in Turbo Pascal, and Tom Swan writes about string handling in Turbo.

Rex Jaeschke, a member of the ANSI Standards Committee on C, begins a series of articles on the ramifications of the proposed ANSI C standard.

The Reader Survey

Keep sending in those reader surveys. *(At the back of the last two issues, postage prepaid.)*

So far we've accumulated a couple fat folders of interesting comments. We'll print results in the next issue if the stream has trickled off by then. I feel a little ambivalent about publishing all the information—it's kind of like an exit poll, or like examining yourself in the mirror, or something like that.

You can bet we'll be trumpeting the results to prospective advertisers, though, because PJ readers are adept and powerful computer users, more so than we guessed. For example, 91% of respondees use their micros professionally, 86% of respondees use three or more languages.

There's one disturbing trend in the comments—a dislike for 'all that UNIX crap,' a dislike for 'that dTACK drivel,' and assertions that we have a 'snide' and 'smug' attitude towards IBM, and are 'always criticizing the IBM PC.'

Is all of this discomfort with 'criticism' of IBM? (*DTACK GROUNDED* often discusses the 68000 family in glowing terms, and UNIX is Bell Labs, not PC DOS.)

Let me make it clear that this magazine is devoted to the IBM, and will remain so. I think that IBM is a superb company, and that it makes beautiful machines. There are people who hate IBM, but they probably hate their spouses too.

It's a fact that the micro world is painfully divided into the Motorola and Intel camps, and that the Motorola CPU, by virtue of its linear address space and high clock speed, has some undisputed advantages over the Intel line. I still remember the chill I felt when I first heard an Intel engineer describe the wonderful advantages of having a segmented address space. I have little doubt that at the time IBM's decision was made, there were some sound reasons to go with Intel, not all of them sinister and Machiavellian. But nobody can pretend that it is fun to program in a segmented environment.

The split is painful because it's hard to get a foot in both camps. You've either got one machine, or you've got the other. Perhaps the most that can be done is to read and write about both sides. I don't think it's good for us—people trying to make a dollar and have fun in the field—to be partisans and just stick our heads in the silicon.

I think that IBM can take the comparisons, I really do. But for those of you who bleed blue, be assured: We're committed to IBM.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Pilgrim's Progress

The computer world sure is exciting. The software project moves through beta test with good marks. Five magazines commit themselves to reviews. The marketing company is named software company of the year by an East Coast magazine. The product is ready for market. How do the marketing company principles react to this good news? Do they throw a party? No, they don't print the manual, fail to finish the hardware, begin suing each other, fire each other, lock each other out of offices, call the police. Oy Vey! Probably just a routine week at the office.

DTACK

I've enjoyed our *DTACK GROUNDED* excerpts, and I'm sorry to see the newsletter come to an end. Hal Hardenbergh has gone on to writing for the Atari 520 ST. We've solicited work from him, and we hope to hear from him, but the steady outpouring that has filled PJ FORUM will soon be no more.

Maybe some of you who have been soaking up PJ FORUM can make the sacrifice of trading some writing for our small honorarium. Our author guidelines: Write about what you know, and put it in a form we can read. We're very strict about that.

Peer Review

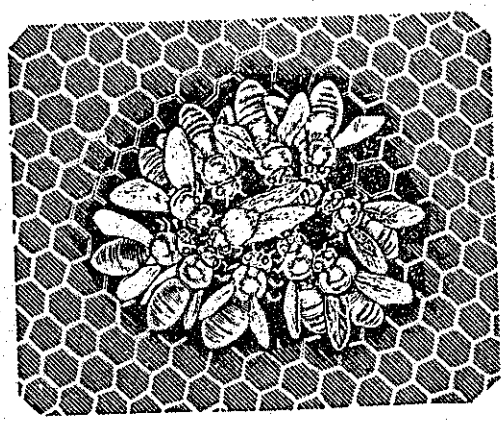
W. Herron of San Diego has suggested a ***peer review card*** in each issue with which readers can rank articles. Done. You'll find the first card, postage prepaid, in the Housekeeping section at the back of the magazine. Beginning with the next issue, we'll be posting the results.

Bulletin Board

We're seriously considering a bulletin board. Any suggestions?

—Robert ☐

programmers' journal
PRESENTS
☆ Amazing Technical Experiences ☆



#1.
A look deep inside the B: drive.

FREE!
UNIX for the PC!
(WITH ANY PURCHASE OF $250 OR MORE BEFORE 4/26/86)

C
DYNAMO

LIBRARY: *325 fully tested functions screen handling/graphic, cursor/ keyboard/data entry, string handling, status and control, utility/DOS/BIOS/ time/date, printer control.*
4 disks, 400 page manual $129.95

WINDOWS: *The most powerful windowing package available, overlays, borders, pop-up menus/ help windows, zap instantly on/off screen, status lines, horizontal/ vertical scrolling, color control/ highlighting, word-wrap, files to windows, keyboard to windows*
3 disks, 225 page manual $129.95

SUPERFONTS FOR C: *Dramatic, high impact screens. Use our character & image libraries and functions, or create your own.*
Font & Function Library $49.95

B-TREE LIBRARY: *Fast indexing B-tree. 16 million each: keys and records, unlimited keys per file, variable length records. PC-DOS, UNIX, XENIX, CP/M.* $79.95

ISAM DRIVER: $49.95

COMPATIBILITY: *PC/XT/AT; Full K&R, CI-C86/De Smet/ Lattice/ Microsoft 3.0/Aztec/others.*

VALUE: *All source code, No royalties. Best documentation available.*

No matter what else you have, you need these.

C-TERP: *Simply the best C interpreter available. See C Journal Summer 1985*
C-TERP (specify compiler) $299.95

PC-UNIX: *Multi-tasking, networking multi-user. With source code* $99.95

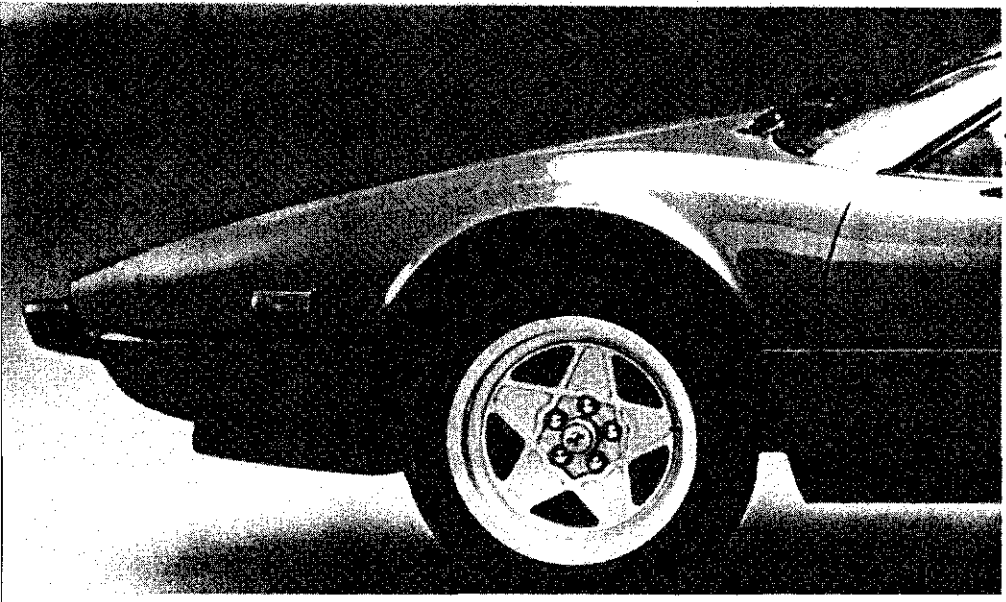
COMBINE AND SAVE!
C LIBRARY plus C WINDOWS
BOTH for only $179.95
+ SUPERFONTS FOR C . $199.95
+ B-TREE and ISAM $299.95
(A $440 VALUE)
C LIBRARY plus C WINDOWS
+ SUPERFONTS
+ C-TERP (a $810 value) $459.95
All (a $740 value) $549.95

Entelekon

12118 Kimberley, Houston, TX 77024
713-468-4412
VISA MASTERCARD CHECK COD

Circle Number 9 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



C Programmers! May PforCe Be With You.

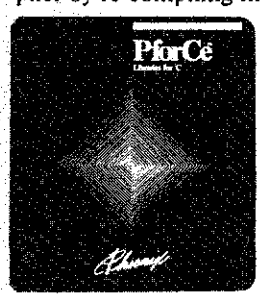
Writing in C? Half your job could be done already. With PforCe.™ The first library of object-oriented C functions and subsystems. Written in C. Fully integrated, optimized, debugged, and ready to go.

High level functions that let you manipulate objects like windows. Fields. Screens. Menus. Change an object's characteristics globally. Or tailor them to a specific application. So you can write code faster. And more economically.

Low level functions to give you complete hardware control. Defaults you can change at will. Plus, more sophisticated subsystems to handle complex tasks. A database system with demand paging and B-trees to store, access and index data. And, since PforCe includes source code, you can modify anything in the library.

But that's not all. PforCe is more than a program-generation toolbox. It's designed to make things easy to find and use. Alphabetically. By functional group. Or, while you're editing through a pop-up utility. And PforCe comes with an easy to follow tutorial to help you become more productive quickly.

PforCe is available for Microsoft,® Lattice,® CI86,™ and Wizard™ compilers. All memory models of each compiler are supported. You can use PforCe with any supported compiler by re-compiling the source code, but we provide a pre-compiled version for the compiler that you specify at order time. $395 complete.



Send for your PforCe™ information kit today. Call or write:
Phoenix Computer Products Corporation
320 Norwood Park South
Norwood, MA 02062
(800) 344-7200.
In Massachusetts (617) 762-5030

Programmers Pfantasies™
by

Phoenix

PforCe is a trademark of Phoenix Computer Products Corporation
Microsoft is a registered trademark of Microsoft Corporation
Lattice is a registered trademark of Lattice, Inc.
CI86 is a trademark of Computer Innovations
Wizard is a trademark of Wizard System Software

Circle Number **18** on the Reader Service Card

More on Variant Records

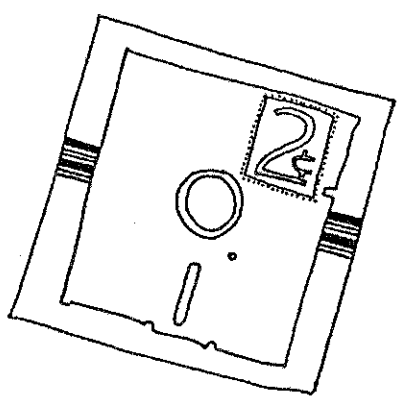
Regarding the letter in *PJ*, Vol. 4, No. 1 entitled "VARIANT RECORDS," *de gustibus non disputandum est.* I'm well aware of the TURBO 3.0 manual's fancy "regpack" type definition—I frequently have to field questions from people who are mystified by that very thing. The case in point is not merely a variant record, but what is called a "free union." An ordinary variant record (called a "discriminated union") would have a "tag field" to determine which of the variants was to be used. To quote the TURBO manual:

> The use of free unions is infrequent and should only be practiced by experienced programmers.

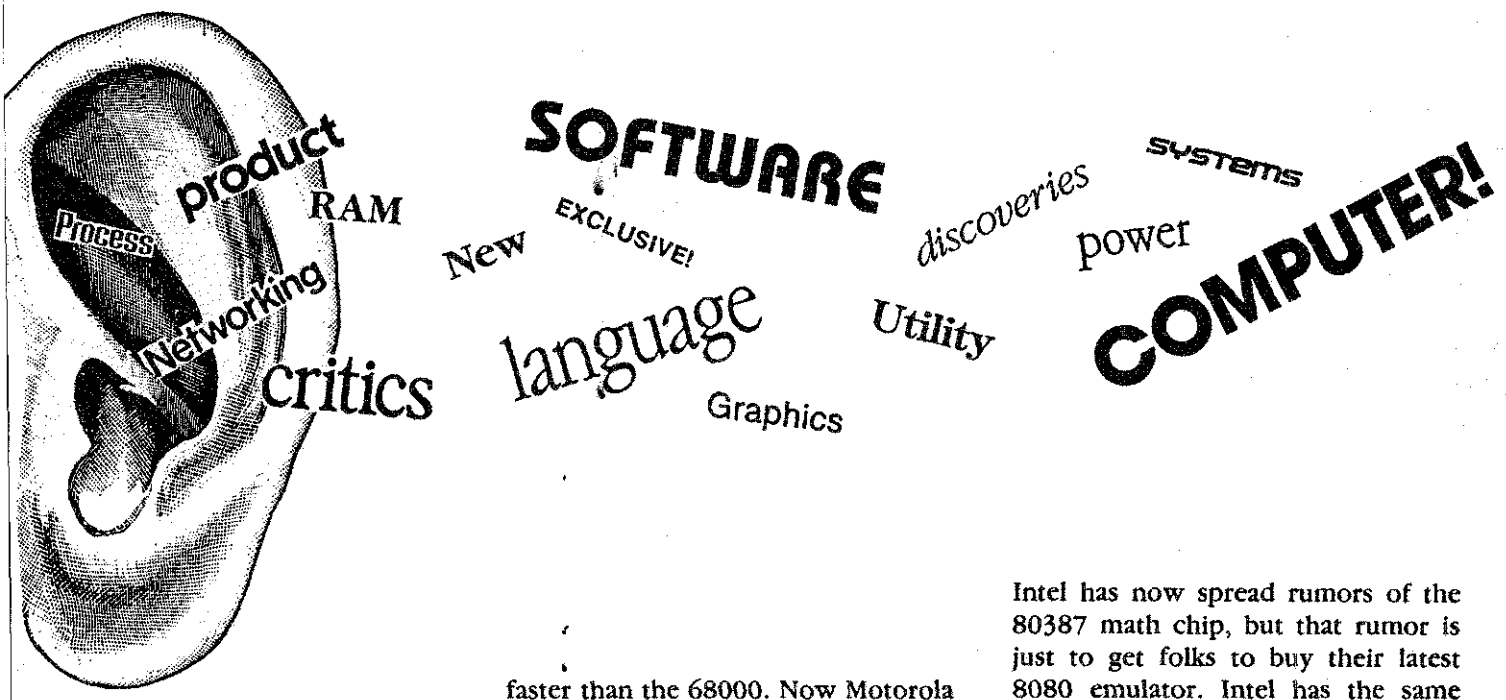
In general, I prefer not to encourage use of that particular technique—simply an aesthetic decision.

I can't help a faint resentment at Mr. Graves' suggestion that I haven't read the manual. SO I have no compunction about pointing out a large error in the sample variant record contained in said letter. REAL numbers in TURBO take 6 bytes, not 4. This is stated repeatedly in the manual. If anyone has doubts, just run this little program: "begin WriteLn(SizeOf(real)); end.". The number "6" will be printed loud and clear.

Neil J. Rubenking
San Francisco, CA 94102 □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



THE INTEL 80386, SINGLE-VS. MULTI-USER, MULTI-TASKING COMPUTING, BENCHMARKS

Excerpts from DTACK GROUNDED: The Journal of Simple 68000/32081 Systems

by FNE, © 1985 by Digital Acoustics, Inc.

Farewell to Dtack:

DTACK GROUNDED the newsletter, that is. We have decided to take the abundant advice we have been freely dispensing and enter the mass personal computer software marketplace. The product will be a very fast text-oriented interactive BASIC for Macki, called HBASIC. It will be written to win in assembly. That means that Digital Acoustics can no longer afford to permit its best programmer to expend 60 hours a month working on this newsletter. . . .

68020 vs. 80286:

Many of you have noted the entirely factual ad by Intel which proves conclusively that the 286 is 2 or 3 times faster than the 68000. Now Motorola is running an ad which shows a white knight beating the hell out of a black knight. The white knight bears the 68020 programmer's model of lots and lots of general-purpose 32-bit registers on his shield; the black knight bears four (4) each 16-bit registers on his. Oh, yes: Bar graphs are included which conclusively prove that the 68020/68881 is up to 6.7 times faster than the 286/287.

We have begun to suspect that the Intel ad and the Motorola ad cannot both be correct . . .

Intel Watch:

While the 386 design started out looking a lot like the 68020, that was a few years back. There then ensued, as we told you a long while ago, a ferocious fight between the Intel folk who wanted the 386 to be a real computer and the folks who, noting that not one of the numerous warehouses containing 8080 code had burned down, wanted the 386 to be yet another 8080 emulator like the 8086, 186, and 286. And, as we told you a long while back, the emulator folk won that fight.

Intel has now spread rumors of the 80387 math chip, but that rumor is just to get folks to buy their latest 8080 emulator. Intel has the same problem everybody else in the chip business has; there are just so many skilled hands available to lay out those VLSI parts. As the 386 approaches first silicon, the available skilled hands are moving on, NOT to the 387 math chip, but to Intel's first non-8080 emulator chip. It is a RISC machine (in fact, it is probably IBM's RISC machine). Surprise? No, the real surprise will be if the 80387 sees first silicon before 1989. In fact . . .

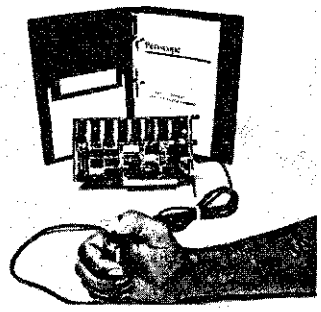
The Intel 386 Previewed:

The Intel 386 has added no additional general purpose registers, but has extended the existing 16-bit registers to 32 bits. The assembly syntax to access the 32-bit version of the AX register, for instance, is LAX, and the 16-bit SP register becomes LSP.

Most of the 386 instructions are identical to 286 instructions and execute (on average) in the same number of clocks. The added instructions are 32-bit multiply and divide and some bit-field instructions a la the 68020. Intel claims that the 386 is 2 to 3 times faster than the 286, which means they are comparing a 16MHz 386 against an 8MHz 286. The 386, like the 68020, is scheduled to be a 12 or 16MHz part.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Periscope Delivers Professional Debugging Power



Models I and II are the same except Model I includes a write-protected RAM board. If you buy Model II, then find you need the board's protection, you can trade up to Model I. Both models include the break-out switch.

GET YOUR PROGRAMS WORKING FAST! "A problem we had for three weeks was solved in three hours," says Wade Clark of MPPi, Ltd. The break-out switch lets you stop your system anytime, even when it's hung, to see what's going on. Symbols, plus source debug for high-level languages, save you hours. Powerful breakpoints on registers, memory, I/O ports, and code help you find bugs fast.

MAKE YOUR SOFTWARE RELIABLE! "I can't live without it!! BRIEF, a text editor my company wrote, would not be as stable as it is today without Periscope," writes David Nanian. Periscope helps you find ALL your bugs so your software is reliable. The latest release includes traceback, which shows up to 2,01o previous instructions, EGA support, user exits, text file viewing, 8087/80287 status display, and much more.

DEBUG PROGRAMS OTHER DEBUGGERS CAN'T! Debug memory-resident programs, keyboard routines, interrupt-driven programs, device drivers, and DOS. Periscope gives you the best crash recovery capability available. "Periscope is rock solid," says user Doug Roberts.

GET THE BEST VALUE! Time and again users tell us Periscope pays for itself in a few hours after they begin using it. This means no professional software developer can afford NOT to try it!

PERISCOPE I: Board, Switch, Software, Manual, Reference Card ..... ONLY $295

PERISCOPE II: Switch, Software, Manual, Reference Card ..... JUST $145

PERISCOPE REQUIRES: An IBM PC, XT, AT or close compatible; DOS 2.0 or later; 128K RAM; one Disk Drive; and an 80-column Monitor.

Call Now, Toll-Free, For More Information Or To Place Your Order

800-722-7006

30-Day Money-Back Guarantee

Get Your Programs Up and Running;

UP PERISCOPE!

Data Base Decisions • 404 | 256-3860
14 Bonnie Lane • Atlanta, GA 30328
Circle Number 6 on the Reader Service Card

The on-chip memory management provides for both segmented and paged virtual memory. The segmented memory management is compatible with the 286. The paged memory management is fairly rudimentary, with only two levels of indexing provided due to space limitations. The 386 has about 300,000 equivalent transistors and measures 1 cm on each side.

The prefetch queue has apparently been expanded beyond the 8 bytes of the 286, because the 386 takes longer to perform a branch. A micro spends 25% of its time branching, and the prefetch queue has to be flushed at every branch.

THE 386 DOES NOT HAVE AN INTERNAL CACHE LIKE THE 68020! It seems that if you include both segmented and rudimentary paged memory management on-chip, there isn't room left for a cache.

The 386 includes a barrel shifter to speed up 32-bit multiplies and divides, although they describe the barrel shifter as "proprietary circuitry." We believe the "proprietary circuitry" tests for a powers-of-two multiplier, in which case a special microcode sequence is invoked to speed up the multiply. Execution time for 32-bit multiply is said to be .6 to 2 microseconds.

Intel is already providing limited-circulation performance comparisons of the 386 vs. the 68020. You already know that the 386 is going to win, right? FIRST LET US DESCRIBE THE MODELS ON WHICH THOSE COMPARISONS ARE BASED:

The 68020 has a 256-byte internal cache. Access to this internal cache takes two clocks/access. The 68020 is assumed to have a zero-wait-state external cache memory, probably a 4K byte one made up of fast 1K X 4 static RAMs. Three clocks (180nsec at 16.67MHz) are required to access the external cache. Finally, Intel assumes that the main memory (DRAM) access will require seven clocks for the 68020 (420nsec at 16.67MHz).

The 386 has no internal cache, just an 8-byte or so instruction prefetch queue. An external zero-wait-state cache is assumed, with two clocks required per access (this is faster than the three-clock 68020). Two clocks at 16MHz is 125nsec. Main memory (DRAM) access is assumed to be six clocks (375nsec at 16MHz).

(We think Intel is underestimating possible main memory DRAM access speed, but we are not sure. After all, there is the extra complexity of that external static-RAM cache. . . . In any event, this is a neutral factor which would affect both the 386 and the 68020.)

To compare the performance of the two systems, you have to make assumptions about the "hit rate" of the cache(s). Intel's comparisons assume a 90% cache "hit rate" for both systems and then for the 68020 further break this figure down for the internal and external caches. Intel asserts that the "hit rate" on the two-clock internal cache is only 12.5%, while we are told the true figure is closer to 40%. (The "hit rate" is highly dependent on the type of software being run. Assembly code commonly features many small loops which fit entirely within the cache whereas HLL compilers do not commonly produce code with many small loops. So assembly-based programs will, as usual, run much faster on a 68020.)

Intel also compares clocks-per-instruction, using the old RISC trick of comparing simple 386 instructions against complex 68020 instructions. *Never mind that it would take more than one 386 instruction to perform the same task as the single complex 68020 instruction!* Well, Intel wants to WIN this comparison, right? . . .

We are betting that the 68020 will out-perform the 386 at equal clock speeds. And remember what fits the 286 gave (and is giving) Intel before reaching its predicted clock rate? But the big reason we believe that the 386 will be slower is that Intel is still carrying around the architecture of the 8080 and its too-few registers, while Motorola has already made the need-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ed clean break from the 8-bit architectures. That is the kind of fundamental consideration which no amount of jawboning can paper over—even though Intel tries!

We are prejudiced, since we make 68000 stuff. But you knew that, right?

Let the Good Times Roll:

Boy, can things turn around in a hurry! As recently as one month ago watching the personal computer industry was about as exciting as watching them hose down the Post Office parking lot on Friday evening in Beaverton, Oregon (that being the big event of the weekend in Beaverton). Watching IBM steamroller the personal computer industry, three-piece-suit division, has been like watching the Nazi blitzkreig roll over the low countries. No fun at all.

Central to this turnaround (in our perception, at least) is the fact that Atari is shipping the Jackintosh! . . .

But the excitement is by no means confined to Jacki. While we were waiting for the super-secret 32-bit CMOS RISC machine from Fairchild, a totally unexpected contender has emerged from England. Now we will begin to see some real competition between not just CISC and CISC (68020, 32032, 386, Z80000) but also between RISCs and CISCs AND, get this, AND microprocessor RISCs and RISCs! Our (us microprocessor freaks') cup runneth over! And best of all, from our point of view, all that competition is going to bring the 68020/68881 combo price tag down more rapidly than would otherwise be the case.

In fact, the 68020 is now in full production in its 12.5MHz guise. As soon as Motorola has its 16.67MHz test station running, we'll see faster parts, too. . . .

A Third Choice?

The following paragraph by Gene Wilburn is quoted from July *InfoAge*, Canadian micro magazine:

Brand New From Peter Norton

A PROGRAMMER'S EDITOR

that's *lightning fast* with the *hot* features programmers need

A WONDERFUL EDITOR FOR PROGRAMMERS FROM PETER NORTON

THE NORTON EDITOR

only $50

Easily customized and saved ▲ *Split-screen editing* ▲ *A wonderful condensed/outline display* ▲ *Structured programming features:* ● *Auto-indenting for* Pascal *and* C ● *A hot "find matching punctuation" feature* ▲ *Great word features too:* ● *Paragraph reformat* ● *Word wrap, word action* ● *You can write your programs* and *your manuals with this one editor* ▲ *Mouse support, too* ▲ *Great for* assembler, *fabulous for* Pascal *and* C

A STEAL AT $50!

Direct from the man who gave you *The Norton Utilities, Inside the IBM PC,* and the *Peter Norton Programmer's Guide.*

"This is the programmer's editor that I wished I'd had when I wrote my *Norton Utilities.* Now I've got it for you to use. You can *program your way to glory* with *The Norton Editor.*"

Peter Norton

Peter Norton, 2210 Wilshire Blvd., #186
Santa Monica, CA 90403, 213-826-8032
Visa, MasterCard and phone orders welcome

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"Users are being conditioned to expect more and more from their packages, either in terms of what the programs actually do, or how they appear on the screen. This can lead to hardware and software designs that actually shortchange buyers. Look at the Macintosh. A snazzy interface and (claimed) ease of use are achieved by frittering away the resources of a Motorola 68000 chip that spends most of its time drawing pictures on the screen. This same chip is often used to drive a 4 to 6-user UNIX computer with multi-user, multi-tasking applications. Which implementation of the chip is paying larger dividends?"

We prefer a third approach, Gene. What say we give all that power to just one user to run just one task? And not fritter away the CPU resources on bit-maps?

Oops! Department:

We are informed that IBM's TopView is NOT bit-mapped, but character-mapped. We knew that it was character-mapped in the basic text version, but thought it became bit-mapped using IBM's new EGA and PGA graphics adaptors. Uh uh.

On the other hand, Topview is apparently even more of a memory-hog than we reported. We understand that if you have a 512K PC, and load TopView, then Microsoft WORD, the PC goes off to never-come-back-from land.

All 17 of you multi-tasking freaks might be interested to learn about a cute development in IBM PC land: Application programs are being written to grab as much memory as they can right away rather than only as much as they need. The reason is that the new multi-tasking operating systems (such as TopView) will lock them into a too-small configuration as more programs are loaded and memory gets filled up. To prevent being locked out in this fashion, the application programs have been modified to grab lots of memory as soon as they are loaded. As a result, loading several application programs under TopView (and related programs) fills memory VERY QUICKLY!

Apparently WORD grabs a bit too much . . .

Computer Design Watch:

Tom Williams starts off the introduction to July's special report on graphics technology by asserting:

"If anything can be said about computers in general, it is that they are continuing to get more 'personal.' More personal means more and more computing power put into the hands of a single user."

Noooo! Really?

Believe it or not, there are some folk out there who haven't gotten the word yet. Wang Laboratories, for instance. Their PC/AT clone was just introduced, and Wang proudly pointed out that where the AT could support only three users, the new Wang clone could support FOUR! One might guess that Wang has not been paying atten-

Atron's PC/AT Bugbusters

Hardware-assisted Software Debuggers for Bullet-proof PC/AT-based Products

A BUGBUSTER STORY

Brad Crain, a project manager at Software Publishing (the people who developed both PFS:WRITE and PFS:FILE), relates the following: "On Friday, March 22, 1985, I was about to get on an airplane with Jeff Tucker, who was co-author of PFS:WRITE with me, and fly to IBM's Boca Raton, Florida facility. For a week, we had been unsuccessfully trying to isolate a bug in a new software product. In a last, desperation move, I set up an early-Saturday morning appointment with ATRON.

"Three of us walked through ATRON's door at 8:00 the next morning. Using ATRON's hardware-assisted debugging tools, we had the problem identified and fixed by 10:30AM."

Mr. Crain concludes: "We'd never have found the bug with mere software debuggers, which have the bad habit of getting over-written by the very bugs they're trying to find. It doesn't surprise me that almost all the top-selling software packages were written by ATRON customers. Now that they've broadened their PC family of debuggers to include a PC/AT debugging tool, those of us seriously into 80286 development are greatly relieved."

ARE YOU TRYING TO DO SOMETHING SCAREY?

Like developing your AT-based software product in the dark? Without professional debugging tools?

Seven of the ten top-selling software packages listed by the *THE WALL STREET JOURNAL** were produced by ATRON customers. The PC PROBE™ bugbuster ($1595) accounts for much of this success. Now that the PC/AT is the new standard for advanced commercial and scientific development, ATRON is proud to announce the AT PROBE™ bugbuster ($2495). It has even more debugging capabilities than the PC Probe.

HOW BUGBUSTERS KEEP YOU FROM GETTING SLIMED

The AT PROBE is a circuit board that plugs into your PC/AT. It has an umbilical which plugs into your 80287 socket and monitors all processor activity.

Since AT PROBE can trace program execution in real time, and display the last 2048 memory cycles, you can easily answer the questions: "How did I get here?" and "What *are* the interrupts doing?"

It can solve spooky debugging problems. Like finding where your program overwrites memory or I/O - impossible with software debuggers.

You can even do source-level debugging in your favorite language, like C, Pascal or assembler. And after your application is debugged, the AT PROBE's performance-measurement software can isolate your application's bottlenecks.

Finally, the AT PROBE has its own 1-MByte of memory. Hidden and write-protected. How else could you develop that really large program, where the symbol table would otherwise demand most of your PC/AT memory.

BORLAND'S PHILIPPE KAHN: "THERE WOULDN'T BE A SIDEKICK™ WITHOUT ATRON'S DEBUGGERS."

So why waste more time reading though your program listing for the ten thousandth time, trying to find why your program starts howling with every full moon. Be like BORLAND, get your Atron bugbuster today and bust bugs tomorrow.

ATRON
THE DEBUGGER COMPANY
20665 Fourth Street • Saratoga, CA 95070 408/741-5900

*WSJ June 24, 1985, reporting Software figures. © 1985 by ATRON. PC PROBE™ and AT PROBE™ ATRON SIDEKICK™ Borland IBM Corp. owns numerous trademarks. Ad by TRBA.

Circle Number 1 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

on to what is actually transpiring in ıe marketplace? Maybe that's why Wang lost $109 million last quarter.

. full-time computer professional ıughingly told us when the DEC Mi-roVAX graphics workstation was in-roduced, "Why, DEC claims that the ficroVAX has nearly all of the power)f a VAX 11/780 but it can't support NYWHERE NEAR as many users!" We have news for that full-time com-)uter professional: Our DEC contact ells us that the MicroVAX is being)urchased just like the AT, meaning or one user and one user only.

We have turned full circle. The minicomputer industry—initially meaning DEC—started because a)unch of weird engineers wanted heir very own computers, without having to time-share the mainframe. The accountants never have, and never will, understand that attitude.) A lot of time passed and guess what? DEC, which started the whole personal computer game, became a company which sold VAXen for companies to chain lots of engineers to. So the engineers started sneaking PCs in the side door, where the accountants wouldn't notice.

And now DEC is once again selling a PERSONAL computer workstation. We wonder whether DEC did that on purpose, or whether they stumbled into it by accident, the way IBM did with the AT?

Come to think of it, the very first computers were single-tasking devices, and hence were controlled by one user at a time. That means we have traveled the circle twice! First the mainframes became time-sharing because the accountants proved that was more cost-effective. So the minicomputer was invented so us engineers could have our own personal computer. Then the accountants discovered that it was more cost-effective to time-share those minicomputers and the 32-user VAXen were born. So the personal computer was invented for the third time.

We predict that next month the accountants will discover that it is more cost-effective to time-share the $995 Atari 520 ST, aka Jacki. And we will start around the circle for the third time. All of the available evidence indicates that a 7-8 MHz 68000 has over half of the computing power of a VAX 11/780 (not counting the floating point accelerator). Honest.

(If architects were accountants, your FNE would live in a large dog-house instead of a 3200 sq ft hilltop home. How much room does one person need, the accountant would ask?)

Benchmark Stuff:

Otherwise Intelligent translated the assembly benchmark on the back of issue #44, the one which ran in just over 1 second for 10 iterations on a zero-wait-state B.G.+. Run time on his 6.8MHz 32016, with four wait states, was 3.8 seconds. So much for the infinitely superior Nat Semi chip.

An unidentified Motorola type with access to a 12.5MHz 68020 system with 4K of zero-wait-state cache memory plugged in our listing and clocked 0.57 seconds, 10 iterations. That's 1.85 times as fast as our 68000 with the same clock, which means Motorola's assertion that the 68020 is 3.5 times as fast as a 68000 while running 68000 code, with a 2–1 (16 MHz vs 8MHz) difference in clock speed, is slightly conservative. That's no surprise; we have discoverd that Motorola has been pretty straight on benchmark comparisons in the past.

The same Motorola type compiled the C version of the *BYTE* benchmark and ran it in 0.80 seconds, 10 iterations. Look: That is as idiot-simple a benchmark as you are ever going to see. And even with register variables our unoptimized assembly version ran 40% faster. (You are not SUPPOSED to optimize that benchmark; the idea is to see how fast the computer/language combo runs the specified operations.) Whatever happened to the famed C with only 5% HLL overhead?

More C Sieve benchmarks (all for 10 iterations): See Aug *BYTE*, p. 132. We will reprint the results, adding the reported 68020 time:

8088	4.77MHz	11.6 sec
80286	6 MHz	3.71 sec
VAX-11/750	?	2.41 sec
VAX-11/780	?	1.90 sec
32032	10MHz	1.85 sec
68020	12.5MHz	0.80 sec

The 68020 whips the 10MHz 32032 by a factor of 2.3–1—both in C—with the 32032 using the fabled Green Hills C compiler, no less! And using register variables, too—see Listing A on page 133. Now we have some real-world comparisons between the 680X0 and the 320XX processors, and the 320XX does not match up well, either in assembly or in C. No wonder the Nat Semi ads emphasize the completeness of the chip set rather than performance. Performance-wise, it appears that Otherwise Intelligent has been gulled by the folks at Nat Semi.

The Nat Semi 320XX does have a beautifully orthogonal instruction set. Not nearly as efficient as the 68000 instruction set, but BEAUTIFULLY ORTHOGONAL!

On p. 122 of Aug *BYTE*, Phillip Robinson is pleased that Nat Semi was able to start on the 320XX series from scratch, not being burdened down with prior microprocessor architectures and instruction sets to support (a la Intel). Do you suppose somebody ought to tell Phillip that Nat Semi in fact introduced the first 16-bit processor, way back in 1974? And three of the first four 16-bit processors in the industry? And that they all failed? And that even Nat Semi's 8-bit processors failed too? And therefore the reason Nat Semi was able to start with a clean state was that all of its prior microprocessors had flopped?

More Benchmark Stuff:

As many of you readers are aware, your FNE is really stupid when it comes to high level languages, among other things. So we really do appreciate it when we get a half-dozen letters from persons with computer science educations protesting our assertion that the Microsoft compilers are written in C and therefore are ludicrously slow. The letters politely asserted that we were confusing compile speed

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

with run time speed. OF COURSE! We are OFTEN confused!

The Aug *IBM PC Tech Journal* lists some Sieve benchmark times for IBM PC COBOL compilers. The times are for ONE iteration, in which case our 68000 assembly benchmark clocks about 0.1 second and HALGOL about 2.0 seconds. Page 114 lists times for both decimal and binary arithmetic; we will discuss only the faster binary times. The fastest compiled COBOL time was clocked using the REALIA compiler: 3 seconds. The same COBOL benchmark with compiled Microsoft code clocked in at 16 minutes and two seconds!

DRI's Level II COBOL, which is also written in C, clocks in at 30 seconds for one iteration—only 15 times slower than HALGOL, which is an interactive language, and only 300 times slower than 68000-based assembly. On the other hand, DRI's C-based COBOL generates code which runs 32 times FASTER than the code generated by Microsoft's C-based COBOL compiler.

That means the code generated by Microsoft's C-based COBOL compiler for the PC runs 9,110 (nine thousand one hundred and ten) times slower than 68000-based assembly and 480 (four hundred eighty) times slower than HALGOL, which is an interactive (not compiled) HLL language.

Since the PC's 4.77MHz 8088 is about 6 times slower than our 12.5MHz 68000, that means that Microsoft's COBOL compiler implementation, when running binary (not decimal) arithmetic, is 80 times (480/6) less efficient than interactive HALGOL. Say . . . you don't suppose Microsoft's COBOL compiler was written by a superbly-educated computer science type, do you?

Your stupid FNE is grateful to the brilliant computer science types for setting him straight regarding the performance of HLL compilers which are written in HLLs. (No honest!)

(We have a strong hunch that the Microsoft COBOL "compiler" compiles to a sort of p-code which is INTERPRETED at run time. THIS IS A COMPILER?) □

SUPER-CHARGE your directories for only $39.95

Say goodbye to path names and unleash the full power of DOS on your IBM Personal Computer. With **SuperPATH™** from Martin Scot, you can run any program from any directory, from any disk, anytime.

SEE THE BIG PICTURE.
Tired of playing hide and seek with your files? **SuperPATH** tells you where your files are, instantly. Anywhere on your disk.

EXPLOSIVE PERFORMANCE.
Program your directories. To display just the files you want to see, in any format, sorted or double sorted. To set up your keyboard macros for each new directory you enter. To back up your files automatically when you leave a directory.
Or, you can use **SuperPATH**'s batch building option to create your own file management utilities.

NO RISK.
Comes with a 30 day money-back guarantee.

CALL (206) 527-9605 to order SuperPATH.

MARTIN SCOT
Development Corporation
4515 Purdue N.E. Seattle, WA 98105

Circle Number 27 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The FASTEST, EASIEST, MOST POWERFUL BASIC EVER!

Amazing New Advance – ments for an Old Friend.

ZBasic is an incredibly advanced and powerful BASIC—but—it's still the old BASIC you're used to. Instead of spending 6 months of your life learning another complicated language, let ZBasic put your programs into light-speed, now! (If you know BASIC, you know ZBasic.)

How Fast is ZBasic?

Lightening fast. Four years of intense development have produced the ultimate BASIC. ZBasic is "Compiled BASIC," and generates stand-alone applications that make any other BASIC completely obsolete. Just look at these speed comparisons.

ZBASIC Compiler	**13.7 seconds**
Turbo Pascal Compiler	**14.1 seconds**
BASIC Interpreter	**2190 seconds**

Sieve from BYTE, Jan., 1983 on an IBM PC, 10 iterations.

Compiler Speed/Interpreter Ease.

Like a BASIC interpreter, ZBasic allows you to write and execute your programs immediately! No messy "Linkers," "Loaders," or clumsy "Subroutine Packages" like most other compilers. To compile and edit, simply type "RUN." Debugging works the same as the interpreter, too. Just type "BREAK" or "CTRL C" to get back to the editor.

Lightning-Fast Compilation.

Computer Language Magazine says, "Compilation is amazingly fast..." After typing "RUN," ZBasic compiles your program at blinding speed—40 lines per second.

Works the Same on All Computers.

If you're tired of throwing away your old programs everytime you switch to a new computer, ZBasic is for you. Source code is portable from one computer to another, and since ZBasic uses Device Independent Graphics and Disk File commands, your programs automatically "Adapt" to any other computer. And the ZBASIC editor is the same on all versions—regardless of the computer.

Einstein Math.

ZBasic offers programmers a math package that surpasses anything else in the industry! (Yes, ZBasic is even better than FORTRAN, PASCAL, MODULA-2 or any other language available!) You will have up to 54 digits of user-selectable accuracy at your power.

"Superb Documentation!"

"The 387 page ZBasic manual is a model of clarity and organization. The documentation is superb, solidifying our impression that someone worked incredibly hard to make ZBasic a benchmark for all other BASIC Compilers."
PC WEEK, Nov. 12, 1985

Easy Structure—If You Want It.

ZBasic helps you "Structure" your programs in a way that's easy and simple...you may use GOSUB or GOTO with names or line numbers. Supports multi-line LONG IFs and LONG FNs. LIST programs with—or without—line numbers! ZBasic automatically indents loops and structures in LISTings, too.

"Awesome! It's about time!-Great! Unbelievable!" J.R. CPA, Seymour, MO

"...fast, generates stand alone programs, requires only modest amounts of memory, has outstanding compilation speeds and...was bug free and felt solid. And the price is very attractive." Bruce W. Tonkin, COMPUTER LANGUAGE

"ZBasic is a powerful offering for BASIC programmers. It provides the flexibility of Turbo Pascal and the speed of compiled BASIC, all at a price that can't be matched. Kudos to Zedcor and to all users who make wise decisions to use ZBasic to the fullest." Garry Ray, PC WEEK

"The best I have ever seen. I love it! You should be proud of this product." R. R. Manager, Mesa, AZ

"Mind-blower! Easily the best BASIC I've ever seen!" J. D., Baltimore, MD

Special MS-DOS Features:

- Works on IBM PC, XT, AT, JR, TANDY 1000, 1200, 2000, COMPAQ, COMPAQ DESKPRO and most MS-DOS compatible machines. Configures for graphics and text.
- Creates stand alone .COM Files
- Mouse Support.
- Executes DOS Commands, BATCH Programs or other .COM Files from within ZBasic

ONLY ZBASIC GIVES YOU THESE FEATURES:

- **Highlights errors...makes debugging easy!**
- **Not Copy Protected**
- **Never any Royalties or Runtime fees for programs you sell.**
- **Direct commands (Speeds logic testing like an interpreter)**
- **Super Single-Step debug**
- **CHAIN with variable passing. (Share all or some variables)**
- **Create transportable subroutines and functions**
- **Multi-line LONG IF. Multi-line LONG FNs (argument passing)**
- **Decimal, HEX, OCTal or BINary support.**
- **Device-Independent Graphics and File I/O.**
- **Never does String "Garbage Collection"**
- **Comes with "Quick" and "Shell" sort source code**
- **Built in "HELP" screens lets you get answers fast.**
- **Long variable name (15 characters)**
- **Loops: WHILE-WEND, DO-UNTIL, FOR-NEXT-STEP**
- **Serial Port and Modem support**
- ***Easily load your old BASIC programs saved in ASCII***

Versions shipping now!

Macintosh , Apple IIe - IIc (128K & DOS 3.3) •
IBM PC and MSDOS 2.1 & Compatibles •
Kaypro Graphics version (CP/M-80) •
CP/M-80 2.0 or greater (Z80 only) •
TRS-80 Model 1/3, Model 4/4p •

$89.95

Send me ZBasic right away! $89.95 complete.

CREDIT CARD—MASTERCARD/VISA/AMEX/C.O.D.

Card Expiration Date: _____

My computer is a _____

Name _____

Address _____ APT # _____

City _____ State _____ Zip _____

Day Phone _____

MAIL TO: ZEDCOR, INC.
4500 E. Speedway, # 93
Tucson, AZ 85712

Arizona Residents add 5% Sales Tax. In U.S. add $5.00 shipping. C.O.D. add $5.00 per order. Outside U.S. and Canada, add $20.00 per shipment for postage (U.S. currency only.)

ZBasic™

C, Turbo Pascal, IBM PC, Fortran, Pascal, MODULA COBOL are trademarks of their respective companies.—Macintosh is a trademark of Macintosh Laboratories, Inc.

ORDER TOLL-FREE:
1-800-482-4567

30 day money-back guarantee.

Technical Support: (602) 795-3996

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE HOME OFFICE SYNDROME

by Paul Barkley

William Brice, President of Brice Industries, is looking for a computer consultant and has a half dozen names in front of him—names gathered from a local consultant's group and from the yellow pages. He dials the first name.

On the fifth ring an out of breath voice answers, "ABC Consulting Company." The caller replies, "I'm Bill Brice, and I'm looking for a computer consultant. What sort of work do you specialize in?"

"Well, I do a little of everything, I guess. I work on lots of different computers." In the background Bill Brice can hear a baby crying, and a dog barking. Bill asks for a brochure, and it turns out that there isn't one available. ABC promises to send a letter, though, detailing past consulting experience. The conversation is soon terminated.

The next number that Bill calls is answered by a recording. "This is Joe Bloe. Please leave your name and number at the tone."

The third number isn't answered at all, even though it's three o'clock in the afternoon.

The fourth number is another recording, but this one says, "QRS Consulting Company. No one is in the office at the moment, but if you'll leave your name and number, I'll get back to you as soon as possible."

The fifth number is answered, "XYZ Consulting Group, can I help you?" Bill Brice answers that he's looking for a computer consultant.

"Certainly. If you can tell me what industry you're in, or what area of computer automation that you need assistance with, I can direct you to the correct person."

"I'm in the wholesale widget business, and I need a system to track my receivables."

"Mr. Brice, we have several people who can help you. Let me connect you with Mr. Jones."

These consultants all sent a message to Bill—some intentional, and some not intentional. Although there are many things that these consultants could do about the way their phones were answered, most of Bill's impressions were direct results of the types of offices that they maintained.

It was obvious to Bill Brice that the first four numbers belonged to consultants who used a home office. Bill crossed off the number that wasn't answered—this consultant was not only out to lunch, he was also out of the question.

Similarly, the barking dog office was crossed off. That consultant was certainly not working from an optimal environment and probably hadn't been in business too long. The quality of his work was suspect because of his unprofessionalism. Bill doesn't share W. C. Fields' attitude about children and dogs, but he does recognize the signs of disorder.

Joe Bloe didn't sound too bad although Bill would have preferred to hear a company name used. Joe is certainly working by himself and might be consulting only on the side. He probably doesn't have a business license and perhaps not much experience either.

The QRS Consulting Company sounded much more professional to Bill although there was no one in the office to mind the phones there, either. QRS might have been a one or two man firm with no clerical help, but they sounded professional.

The last group, XYZ, was obviously the largest and had the largest pool of talent. The receptionist implied that there were several, and perhaps many, consulting specialists in the firm. This had a lot of appeal to Bill although he wondered if maybe their price might not be higher than a smaller outfit.

I've used a variety of offices as a consultant, and I'll make an inflammatory statement: If you operate out of your home, you'd better do everything you can to conceal that fact because you're fighting an uphill battle. Better clients, in general, demand more professional consultants. Those willing to deal with "irregular" home-based consultants are more apt to run their own businesses in such a manner, and

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

n general, these people make bad clients. They're badly focused on heir own problems and apt to act erratically.

Of course there are numerous exceptions to this, but very few computer consultants can get away with using a home office for very long. Let's look at a number of office alternatives:

1. HOME OFFICE IN AN APARTMENT OR HOUSE. This is absolutely the worst category. Your address on your brochure or letterhead will give you away, and a post office box, if you haven't absolutely established the impression of stability and size, is a sleazy alternative. IBM can use a post office box if it wants, but most companies with no posted address scream "no regular office—we're fly by night."

If a client has to visit your office, to drop off some materials or for a demonstration, you'll be lucky to keep him or her. If you live in a substantially better house than average, though, you can get away with all sorts of home office arrangements because clearly you have enough money to do whatever you want. Some very expensive management consultants have pulled this off quite nicely, but they live in mansions on grounds large enough to be called estates.

2. HOME OFFICE IN A HOUSE WITH SEPARATE OFFICE ENTRANCE. This is much better because clearly your office is not a regular part of your home. Clients can enter it separately, and you signal to them that you're established enough to warrant the segregated office space. (If you live in an apartment above a commercial space, you don't even fall into this category—in the right arrangement clients may not even know you live upstairs.) If you happen to live in the country, in the splendor of rolling hills and forests, this works very nicely because anyone can see your reasons for not wanting a regular office in the city. The more expensive your house is, as above, the more likely you are to get away with this. Your mailing address, though, still has a residential stigma.

3. IRREGULAR SHARED OFFICE SPACE. You may be able to rent a small office in the back of someone else's multi-office suite. Although this will provide you with a business mailing address, visiting clients won't appreciate the fact that you have no sign and that your entrance is clearly through someone else's offices. This is not much better than a home office, and in extreme cases can be worse.

4. REGULAR SHARED OFFICE SPACE. This provides you with a "correct" business address, a proper sign for your entrance, and the use of amenities such as photocopiers, conference rooms, entrance foyers, and secretarial and receptionist services. Your phones will be answered professionally in a business-like manner (in a well-run system), and many seldom used necessities will be available at little cost to you.

While some one-person firms use shared offices to present an inflated, grandiose impression to out of state clients, this impression vanishes when someone visits the office. This type of office gives the impression of a small, but professional company. One large bonus: You may interact, in the hallways and around the coffee pot, with a number of other professionals outside your field. It's always interesting to swap notes, and over the long run you may be able to do business with some of your "office-mates."

5. REGULAR OFFICE SPACE IN AN OFFICE BUILDING. Once you get your own office, you have the problem of staffing it. Although you can use an answering machine or an answering service, there isn't much you can do about UPS shipments that arrive when you're out of the office. You'll be amazed at the number of people who wander through your door when you have a public office. Many will be unsolicited pesky salesmen, but others will be potential clients or those selling something that you need.

If you're often out of the office, though, you'll make a ridiculous impression with one of those little plastic clocks with the movable hands that say "Back at:." In addition, you may spend a lot of time running around looking for your parcels that couldn't be delivered. You really should hire a clerical person to make sure your office is open during all regular business hours. If you choose this arrangement, some interaction with others in your building will go on in the corridors and elevators, and take the time to stop in the offices on your floor and introduce yourself as a neighbor.

6. REGULAR OFFICE SPACE IN A TOWNHOUSE. Many urban areas have a lot of office space in townhouses. This is similar to the previous category, but there's no interaction with other people—your firm is on its own. The smallest space you can take is usually 1000 square feet. This is suitable only for consulting firms that are growing and who want the privacy of a townhouse office.

You Still Want to Work in Your Home

Computer consulting is a risky business, and there are tradeoffs when you consider various office arrangements.

The two advantages to working from a home office are cost and convenience. Not only does a home office not require additional out-of-pocket costs, it can actually save you money in taxes. Be warned, though, that the IRS has some serious reservations about home offices and will very likely audit your return. The IRS wants you to demonstrate that the office space is used exclusively for the conduct of your business—don't try to deduct a corner of the family room, for instance.

I can't tell you how convenient a home office can be. It really feels good to sit down to work while others are commuting through freezing rain or a heavy snow fall—what a luxury! Your productivity increases when you spend that extra hour a day working and not commuting. Your transportation costs will drop, and many people find they can get by on much smaller clothes budgets. (If your accouterments include a plastic pen holder in your shirt pocket, forget

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

this last point—your wardrobe is probably the same whether you're mowing the lawn or calling on a client.)

Unfortunately, besides the serious negative client impression you risk, there are other drawbacks to home offices, too. One problem for some consultants—I'm deadly serious here—is a sudden severe weight problem. The refrigerator is just too handy. Programmers, by nature, are almost all carbohydrate junkies (mea culpa), and stoking the organic furnace too often can have an adverse effect on your health. You may start churning out code at an amazing rate, but your social life will go all to hell, and buying a new wardrobe from Omar the tentmaker can be expensive. This is a description of a pervasive problem, not a random flippant remark.

Distractions are a serious disadvantage for others—children and pets can be substantial interruptions. Some consultants find that although they put in more hours, they get less work done. My advice: Declare regular working hours and stick to them. Shut out your family during these times. At the end of working hours, STOP WORKING and relax. Learn to cleanly separate the two activities. Keep a detailed work log (broken out in 15 minute intervals or smaller), and analyze it to determine how effective your work week really is.

Here's another inflammatory statement: I don't think anyone can be a successful computer consultant on a part-time basis. If you can't afford to give it a shot full-time, then don't plan on making much mark in the world. You won't be able to complete projects except over very long periods of time, and the fact that your efforts will be fragmented will mean that it will probably take you longer in hours spent than it will someone working full-time. If you moonlight for a year or so, you may learn something of the trade, but remember that the time spent consulting will come out of your leisure time—be prepared to give up a lot. Most people are unrealistic about how much they're willing to sacrifice. If you've never worked 60+ hours per week, week after week, why do you think you're going to be able to do that now?

This leads to our final problem with a home office: lack of commitment. A home office is so easy to get into and out of, in terms of commitment, that many consultants give up after putting forth only a half-hearted effort. Remember the classic comment about a ham and eggs breakfast? The chicken was involved in the project, but the pig was committed.

When you commit to an office outside the home, you have a large incentive to keep hustling for work. Although it's possible to get into an irregular shared office space for only a couple hundred dollars a month, regular shared offices in metropolitan areas start at about $500 and run to $700 or $800 very easily. You can bet that you're going to have a very different attitude about collecting your receivables when you know you've got a rent payment to make.

More importantly, you'll be forced to do some financial projections to see whether the amount of business you have will justify the expense. I can't tell you what a bad feeling it is when your receivables dry up and you have to tell your landlord that you're going to be a little late with your rent this month. I had one of my larger, more regular clients stop paying me one time because THEIR main client, the federal government, ran out of money two months before the end of the fiscal year. Needless to say, I'm a little more prudent today about keeping a cash reserve for these unforeseen "disasters."

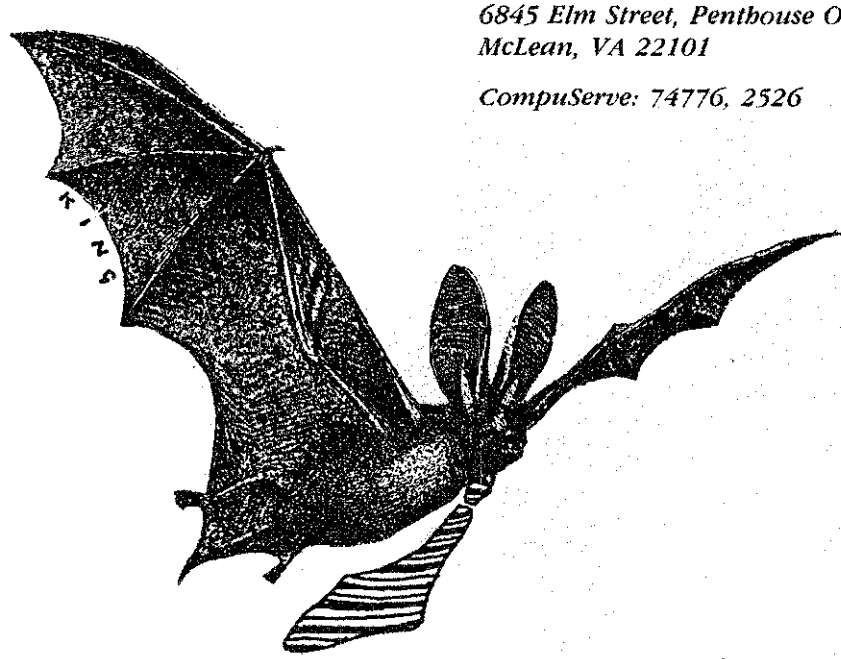
Still, if you're just starting out, a home office may be all you can afford. If so, you'll have to work much harder to overcome the negative stigma. If you fail, it probably won't cost you much—easy in, easy out—but you will have stacked the deck against yourself. Unless you're a consultant who's a quasi-employee (long-term hourly contracts, usually on-site), plan on abandoning the home office as soon as possible.

Are YOU running a successful consulting business from YOUR home? If so, drop me a line and tell me why it works for you, and I'll air your comments in a future column. □

Send your thoughts to

Paul A. Barkley
The Barbican Company
6845 Elm Street, Penthouse One
McLean, VA 22101

CompuServe: 74776, 2526



Auto exec.bat

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

INDUSTRY NEWS

by Frank D. Greco

PC/IX Pact

IBM has signed a 10-year contract with Interactive Systems Corporation that allows the Armonk monster to remarket existing and future versions of Interactive's UNIX and UNIX applications. Recall that Interactive did PC/IX for the PC family and VM/IX for the 4300 family. So you can bet on a System V version of PC/IX sometime in 1986.

IBM has moved its people assigned to UNIX products to Princeton, New Jersey, from Austin, Texas. And they moved the Boca PC group; they are now in Montvale, New Jersey. Brings the recalcitrant child closer to the poised back hand.

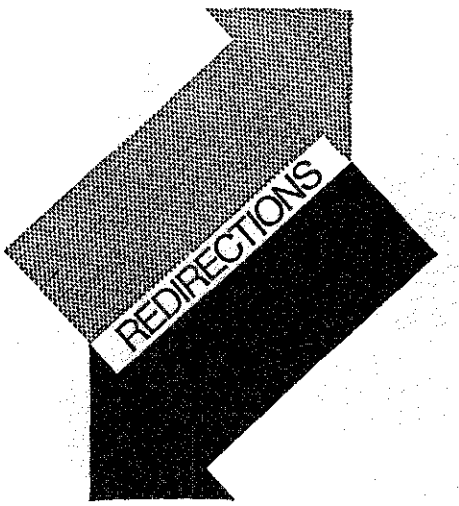
Soda Pop Speaks

The *Computerworld* weekly newspaper (December 9, 1985) quoted Apple President (and former Pepsi executive) John Sculley as saying that all future Apple products will use the AT&T UNIX operating system. This is another big push for UNIX (was there any real doubt friends?). Even longtime UNIX hater Steve Jobs has a company that will specialize in UNIX-based graphics workstations. Dare I say that the rumors about IBM's RISC board for the PC/AT have Berkeley UNIX as the operating system. Chameleons are not the only animals that change color in a pinch.

Pssstt . . . Wanna Hot Machine?

Charles River Data Systems' impressive Universe 32 supermicro uses the no wait state full 32 bit 12.5 MHz 68020. They are advertising it as "maxing out" at 2.7 MIPS. It uses a dedicated 68000 for I/O processing and a 4K byte cache. I have used the dual plain-jane 68000 version of the Universe and was thoroughly blown away by its fast response time with 10 simultaneous C programmers hacking on it. The 68020 version should be a doozy!

Corvus Systems (408-559-7000) is not letting the workstation market pass on by. They have a UNIX-based (what else?) "Cheetah" workstation. It is based on the Motorola 68010, up to 4 Megabytes of RAM, and has a 15 inch monitor that features a resolution of 800 X 620 pixels. It also comes with a 20 Megabyte hard disk drive (no specs available yet) and a 1.2 Megabyte floppy drive. The 1 Megabyte RAM version lists for $ 9,999. Not quite Atari prices, but getting there . . .



The Sord is Mightier than the Stone (age)

Look for Toshiba-owned Sord Computers to come out with a *killer* 68020 version of their UNIX-based microcomputer in mid-86. It will outperform the outdated DEC VAX 11/750 minicomputer. I was told by a very reliable OEM that Sord will start pricing the thing around $5,000! It has a footprint smaller than an IBM PC/AT's. Imagine 10 programmers doing concurrent development on such a small machine. How about 24 end users running a database, spreadsheet, or some other application package. All it needs is PC DOS emulation. I was told that Sord's president was recently in New York City scouring the streets for developers, especially IBM mainframe communications experts.

CP/M at 100 MIPS

RCA Corporation's Advanced Technology Labs in Camden, New Jersey, has built an 8 bit microprocessor from gallium arsenide instead of silicon and has achieved a performance increase of over 100 times that of today's 8 bitters. It runs at 200 MHz and can execute over 100 MIPS (RISC MIPS, that is). Just for comparison, the popular VAX 11/780 superminicomputer peaks out at about 2 to 3 MIPS, and is the size of a large refrigerator. The new RCA processor borrows some technology from the RISC camp in that 90% of its commands are the same bit size, allowing efficient pipelining. Now all that the team at RCA has to do is microcode the thing into either Z80 or possibly 8086 emulation.

Prints and the Resolution

Look for Japan Digital Laboratory Inc. to announce a super high resolution color serial matrix printer in the last quarter of 1985 or first quarter of 1986. JDL Inc. is a subsidiary of JDL Co. Ltd. It was started by the three founders of Advanced Matrix Technology Inc. (see Redirections in *PJ*, Vol. 2, No. 4), who left AMT disgruntled with the vulture, . . . er . . . venture capitalists' approach in market direction. These three former AMT technicians also worked on the old Malibu matrix printers. As I stated in a previous column, the AMT printers are phenomenal. They have the best matrix printer quality that I have ever seen. It seems that their new color printer (JDL) might be perfect for the Commodore Amiga.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Wanna see a Fast Machine? Wanna See it Again?

Recently, a Commodore OEM gave me a thorough demonstration of the 68000-based Amiga. I'm impressed. Technically, it is a superior home machine although it should be priced a little lower for that market. The windowing system is amazingly fast. When you move a window around the screen on a Macintosh or Sun workstation machine, only the outline moves. The window is not redrawn until you release a mouse button and place the window "down."

On the Amiga, the *whole* window is moved around the screen just as easily as moving the mouse (arrow) cursor. This OEM also told me that Commodore is working on a full 32 bit 68020 version. Now that will be FAST! In addition to the speed, don't forget that the 68020 has a memory management unit built on-chip. So unlike the first generation Amiga, which did not have any hardware memory management, a concurrent process on a 68020 version of the Amiga cannot corrupt another process' memory space (unless it has permission to do so).

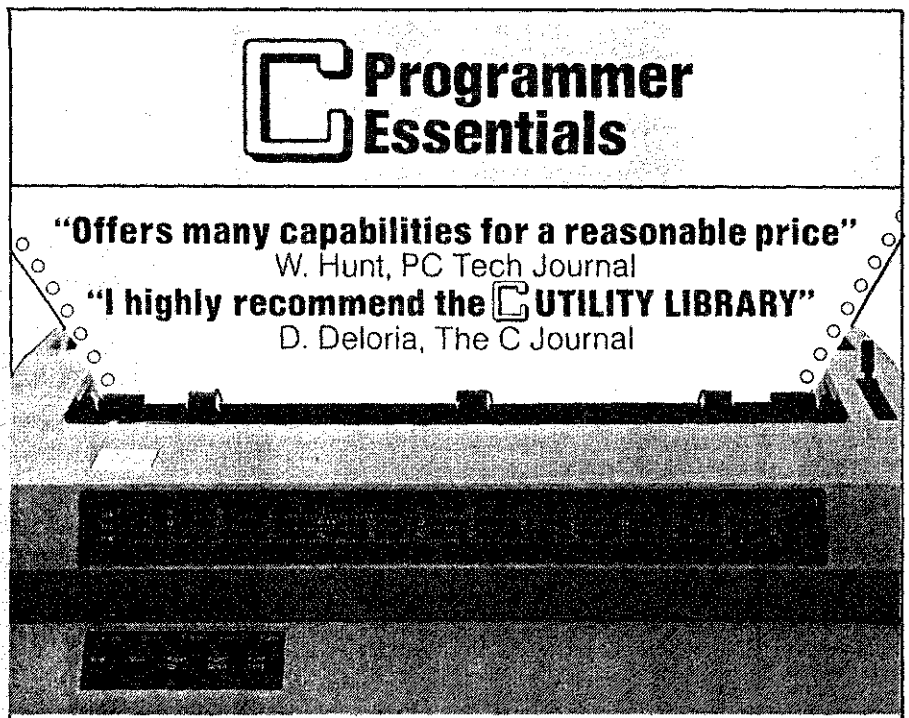
Just as I finished the above paragraph, I found out that Computer System Associates announced a 68020 board for the Amiga at Comdex. UNIX on the Amiga is not far behind. I know of at least two developers currently porting UNIX to the Amiga. Now, that sounds like fun.

More on S1

The OS that chops, cuts, slices and dices is being tested further by Datamedia Corporation and Nippon Univac Information Systems of Tokyo. Multi Solutions Inc. of Lawrenceville, New Jersey, says that Datamedia wants the multi-user, multi-tasking, and multi-processor operating system for its 932 computers. Nippon Univac wants to distribute it in Japan.

Multi Solutions' boastful Chairman Charles J. Lombardo has obtained a minimum 5-year $40 million commitment from Computer Engineering and Consulting Ltd. of Tokyo, Japan. CEC will then have an exclusive license to distribute the operating system in Japan.

Perhaps this is the OS that IBM needs to combat UNIX. IBM hates UNIX. UNIX is the great leveler. Why would somebody buy hardware from IBM when their UNIX software would run faster on someone else's iron? Since IBM is notorious for selling mediocre technology, customers might shop elsewhere. Ironically, AT&T has the same problem on their hands. Since UNIX runs more efficiently on other vendors' machines, commercial customers are not buying the 3B series machines or the 7300 micros (It is a bit weird that the UNIX PC a.k.a. the PC 7300 does *not* come bundled with UNIX). Think of this scenario: If S1 performs as advertised with IBM's support (that's the old proverbial 'if'), it could conceivably put UNIX in the history book. Since it is source compatible with AT&T UNIX (all of the UNIX system calls are implemented in S1), UNIX customers would be guar-


Programmer Essentials
"Offers many capabilities for a reasonable price"
W. Hunt, PC Tech Journal
"I highly recommend the C UTILITY LIBRARY"
D. Deloria, The C Journal

C ESSENTIALS **$100**
200 functions: video, strings, keyboard, directories, files, time/date and more. Source code is 95% C. Comprehensive manual with plenty of examples. Demo programs on diskette. Upgrade to THE C UTILITY LIBRARY for $95.

THE C UTILITY LIBRARY **$185**
Thousands in use world wide. 300 functions for serious software developers. The C ESSENTIALS plus "pop-up" windows, business graphics, data entry, DOS command and program execution, polled async communications, sound and more.

ESSENTIAL GRAPHICS **$250**
Fast, powerful, and easy to use. Draw a pie or bar chart with one function. Animation (GET and PUT), filling (PAINT) and user definable patterns. IBM color, IBM EGA and Hercules supported (more soon). NO ROYALTIES. Save $50 when purchased with above libraries. Available February, 1986.

Compatible with Microsoft Ver. 3, Lattice, Aztec, Mark Williams, CI-C86, DeSmet, and Wizard C Compilers. IBM PC/XT/AT and true compatibles.

C Compiler Packages: Microsoft C - 319, Lattice or CI-C86 compilers -$329. Save $40 - $50 when purchasing compiler and library combinations. Specify C compiler and version number when ordering. Add $4 for UPS or $7 for UPS 2-day. NJ residents add 6% sales tax. Visa, MC, Checks, PO's.

ESI ESSENTIAL SOFTWARE, INC
P.O. Box 1003 Maplewood, NJ 07040 914/762-6605

Circle Number **10** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

anteed of porting their applications to run on a Big Blue box. And IBM would have the upper hand here because S1 is a SUPERSET of AT&T UNIX. That's read as "lock the user into our hardware."

Multi Soft, a subsidiary of Multi Solutions Inc., has entered a joint development agreement with Tata Consultancy Services Inc., a $6 billion company of Bombay, India. The chairman of Multi Solutions, Charles J. Lombardo, said that the two companies will develop and support applications to run on the S1 operating system. Tata Consulting will provide $1 million to Multi Solutions Inc.

Trivial PC Pursuits and Other Pink Noise

How about this one for trivia: The company with the largest number of installed PCs is General Motors with 31,000. Second in line is General Electric with 18,000.

dTAILED dSKETTES

Did you know that Ashton-Tate and Intelligent Systems Corporation both produced their annual reports on diskettes?

A One-ah and a Two-ah and a Tree-ah

The UNIX file system is *not* really a tree structure. Because of the ability of this popular operating system to "link" files, that is, give the same physical file several different alias pathnames, the UNIX file/directory scheme is formally not a tree. Since in a true tree structure "leaves can grow from only one branch," the UNIX file system is strictly-speaking a "directed graph" or an "acyclic graph" data structure.

Ports are Ports

What the microcomputer industry calls "windows" are not true "windows" in a formal sense. Any computer graphics student will tell you that they should be called "viewports."

According to music industry trade magazine, *Billboard*, the number one entertainment software package is Flight Simulator II from Sublogic. It's unfortunately unavailable for the IBM PC. It's listed as being on the charts for a phenomenal 90 weeks. The number one fun package for the IBM PC is F-15 Strike Eagle from Micro Prose, on the charts for 30 weeks. Software listings in *Billboard* magazine? Soon we'll have TV shows like "Software Bandstand" hosted by Dick Clark; programs will be rated by two teeny boppers saying things like "I gave it a 37 because it didn't have a solid feel to it." and "I gave it a 78 because it had a good beat . . . er . . . spreadsheet."

4 out of 5 AI experts agree that . . .

Proctor and Gamble ('better teeth through AI science') has invested $4 million in Artificial Intelligence experts Teknowledge Inc. Lotus has also invested in Teknowledge recently. So has Ford Motors ($1 million) and General Motors and Elf Aquitaine. Hmmmmm . . . something's afoot here and it ain't 12 inches . . .

X-Rated But Still Open

The X/Open Group is a group of European computer manufacturers who are seeking to increase the volume of UNIX applications. The group consists of Bull, ICL, Nixdorf, Philips, Olivetti, and Siemens.

That's too Swede for Big Blue

Something that not too many people know is that IBM's token ring LAN actually was patented by a Swedish engineer, Olaf Soderblum. Two years ago, he forced IBM to pay a large undisclosed cash amount to him to license his LAN. He is reportedly eyeing IBM competitors who are making IBM-compatible LANs and asking them for license fees.

Sort RM/COBOL Files

• • • QISORTC • • •

Finally, a sort utility for RM/COBOL.
Using efficient algorithms in assembly language making it the fastest sort on the MS-DOS market. Qisortc supports:

- RM/COBOL Version 1.5x & 2.xx files
  - Single / Dual File Indexed
  - Relative and Sequential
- ASCII, COMP-1, & COMP-3 fields
- 32 sort control fields
  - ascending / descending
- 32 record selection fields
  - EQ:NE:GT:GE:LT:LE field:constant
  - implied AND, OR may be specified
- 32 output fields or total record
- CALL from RM/COBOL, when provided subroutine is linked with runtime
- Use pathnames and return codes
- Keyboard or Parameter file
- Requires MS-DOS ver 2.xx or above
- Only $95.00 - COD, VISA, or M/C.

Quantumn Information Systems, Inc.
145 N.W. 85th St., Suite 103
Seattle, Washington 98117
(206) 789-2888 Telex 4941041

Circle Number 31 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Speedy Graffiti

Some time ago, when AT&T was making a strong bid to buy into Britain's semiconductor chip manufacturer Inmos Ltd. (303-630- 4279), this column pointed out a reason for the phone company's lust. It's called the Inmos transputer. In short, it's a high-powered, 32 bit RISC-based microprocessor intended to be connected with other transputers to form networks of parallel processors (see Redirections in *PJ*, Vol. 2, No. 4). At the recent ACM Siggraph conference in San Francisco, Meiko Ltd. showed a two-dimensional matrix of transputers connected together to form a graphics system. This "Computing Surface" was built by nine former employees of Inmos (good 'ole America!) and performs at 750 MIPS!! If you think that is impressive, it sells for $150,000. This system consists of 40 boards each with 4 transputers, 48K RAM per board, 2 Megabytes of shared memory, 4 NEC 7220 graphics chips, and the new Inmos color lookup chip (IMSG170). The whole shebang was controlled by another processor running UNIX (there's that word again).

Getting a Little Color

Texas Instruments also showed some color at Siggraph this year. The TMS34070 video chip utilizes digital-to-analog converters, analog interfaces, and a color lookup table to give the user 4 bits of color-palette. This chip will be able to display 16 colors out of a palette of 4,096 at a time on EACH scan line. The lookup table can be changed scan line by scan line, all without the host processor's intervention. So, if you have 240 scan lines and a possible 16 colors per scan line, that's a total of 3,840 possible colors per screen! The first edition of the chip can handle graphics resolutions of 800 by 600 pixels, noninterlaced(!).

Ricoh Ltd. of Japan will soon announce a color, thermal-transfer printer available in America by the end of the year. Preliminary indications price it at less than $400. Seven colors are available, all in 180 dots per inch. Draft quality is 80 cps, near letter quality is 50 cps, and letter quality is 30 cps. The neat thing about this printer is that it will be able to print on plain paper and transparencies.

I Thought We Saw the Last of the Laser Shows in the '70s!

Look for Mita ('mee-ta') Copiers to announce a laser printer engine similar to the Canon device that powers the Apple Laserwriter and the HP Laserjet. It will be initially priced in the range of $500 in OEM quantities. The device is smaller than the footprint of the PC and packs more resolution than the Canon machine. This implies end user laser printer prices in the neighborhood of $1000–$1500 by spring of 1986.

Now that we have laser printers that produce black and white hardcopy, the next things that we'll see are color laser printers. QMS Inc., of Mobile, Alabama, is working with another southern company, Colorocs of Atlanta, to produce a color laser printer by the end of the year. They expect a production model during 1986.

Xerox has recently announced a new laser printer that is directly connected to a high quality copier. Now that's the way you do it.

Money for Nothing, Chips for Free

Look for a redesigned motherboard on the IBM PC/AT within the next 1 to 1 1/2 years. According to some sources in Austin, Texas, IBM is using surface mounting technology to bring down manufacturing costs significantly. IBM is hoping to cut labor costs up to 50%. It would be good if this were passed on to the computing consumer.

Workstation Wars

Keep an eye out for Atarian Jack Tramiel to start hawking a National Semiconductor 32332 based workstation by either the end of the year or the first half of 1986. Reports have it running the Berkeley 4.2 version of UNIX. Valley rumors have Jack tentatively pricing the UNIX workstation (there's that word again!) around the $2,000–$4,000 range for sure. Remember that Steve Jobs wants to compete with Atari with his Next Inc. company. They too will manufacture large-screened workstations specifically for the university market.

Solar Powered Workstations

Rumors whispered at the Sun Microsystems booth at the June Usenix conference in Portland suggested that Motorola has 25 MHz versions of the 68020 in Beta test. "If you put your ear next to the chip, you can hear it whine" was the hyperbole that I heard. These same Sun representatives strongly suggested that Sun may have an Alpha version of a 32 bit 25 MHz workstation in their own labs.

Spitting Bits at 10,000 Baud

DCA/Telebit's (404-442-4000) new Fastlink 10,000 baud modem is able to operate over regular switched-network telephone lines. This new modem, available for the IBM PC in external or internal models, uses two on-board processors: a Motorola 68000 microprocessor and a Texas Instruments TMS 320 signaling device. Perhaps the promise of Videotex can now be fulfilled. High resolution graphics can be sent in the blink of an eye. The board version of the modem comes with Crosstalk-Fast, which is a special version of Microstuf's Crosstalk XVI software. This software also emulates the IBM 3101, DEC VT100, Adds' Viewpoint, and Televideo's 910/920 terminals. Guess what happens to the multiplexor market now that we have modems that are faster than 9600 baud? They are just as popular as the new Saturday Nite Live cast.

All Hands Off DEC

Why hasn't Digital Equipment Corporation really made it in the personal computer arena with their Rainbow PC? Many analysts have their own opinions. Just looking at their latest

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DEC System Expansion Products catalog, I came across the following: 64K RAM chip set, $149; two 5 1/4 inch floppy diskette drives, $995; 5 Megabyte Winchester drive, $750. With these prices, I wonder why they haven't succeeded?

DEC Stops at the Desk Top

Speaking of DEC, while the MicroVAX II with its 125,000 transistor 78032 chip set has already been announced and making mucho buckos for Digital, we can expect an encore in 1986. The MicroVAX III will be announced in late 1986. It will have a 1.22 million transistor microprocessor set (8 chips) that will run the *complete* VAX instruction set in hardware. The MicroVAX II runs only 70% of the VAX instruction set in hardware; the rest is emulated in software . . . sounds like a half-hearted RISC chip! The desktop MicroVAX III will replace the outdated VAX 730/750 family.

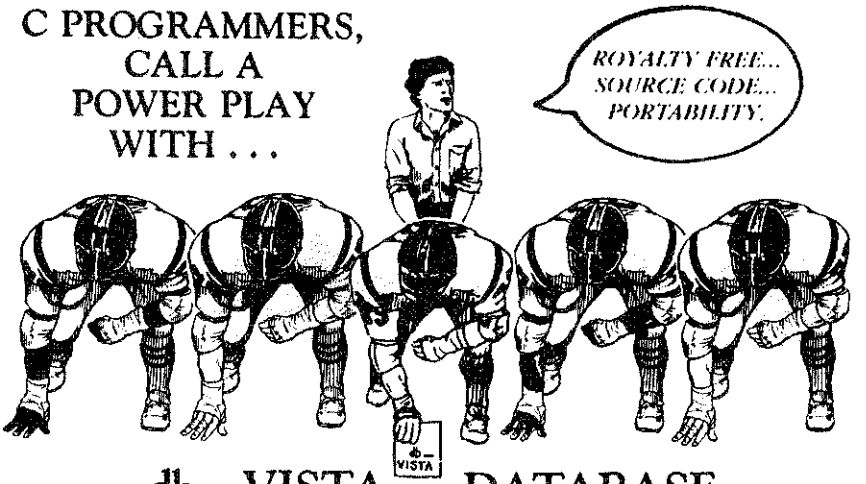
IBM 370? . . . Yeah, I Saw One at a Museum

Wonder how come the IBM AT/370 is not really flying as high as it should? Performance. I have been told by some colleagues who have the machine that it's extremely slow and cumbersome. No wonder—the complex 370 architecture is not exactly easy to miniaturize. Studying the architecture of the 370 in my undergraduate days was the most tedious and boring undertaking that I can remember. Perhaps IBM's old philosophy of being all things to all people (PL/I and VM) is coming back to annoy Big Blue.

Low, Low Prices

I noticed an ad for hard disks recently from Orange County Computer in California (714-898-2943). They had really low prices. Their 10 megabyte hard disk drives are as low as $275, and their 20 megabyte drives are $385. Those are flea market prices! By the time you read this column, they probably will be cheaper!

C PROGRAMMERS, CALL A POWER PLAY WITH . . .



. . . db__VISTA DATABASE MANAGEMENT SYSTEM FOR C.

db__VISTA is a full-featured programmer's DBMS. It handles your data powerfully, yet economically without the frills that make end-user DBMS's bulky, slow and expensive to license. Use only the features you need for maximum efficiency with minimum code and effort.

Powerful lineup of features. B'-tree indexing, multiple key records, transaction processing, interactive database access utility, and file transfer utilities for dBASE, R:base and ASCII files. You even get 90 days applications development support free of charge.

Define your playbook up front. As a network model DBMS, **db__VISTA** is suited to applications development. A premium is placed on efficient use of disk storage, reduced data redundancy and fast access times allowing you to cross the goal line first.

It's your game plan. The database structure is specified by you in **db__VISTA's** data definition language (DDL). The DDL processor compiles the specification into tables (data dictionary) used by **db__VISTA's** library functions, which are called by your C program to manipulate and access the database.

db__VISTA's written in C so you can understand the signals. Source code is optional. No sweat. No royalties, either.

All this delivered for less than the price of season tickets.

Single user without source . . $195
Single user with source $495
Multi-user without source . . $495
Multi-user with source. $990
Available for most popular C compilers under **MS-DOS, XENIX,** plus most **UNIX** systems.

Go for the power play and order **db__VISTA** now. Call (206) 747-5570 or

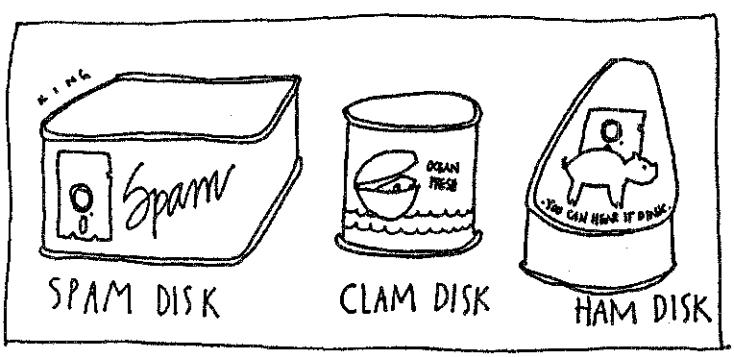
1-800-843-3313

at the tone touch 700-992. 30 day money-back guarantee.

RAIMA CORPORATION

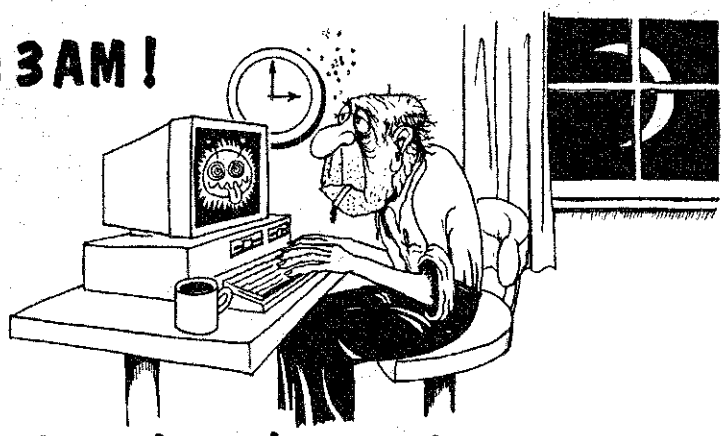
12201 S.E. Tenth Street
Bellevue, WA 98005 USA
Telex: 9103330300

Circle **19** on the reader service card



THE RAM DISK ALTERNATIVES

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Do you know where your bugs are?

This C programmer is finding his bugs the hard way...one at a time.
That's why it's taking so long. But there's an easier way. Use

PC–Lint

PC–Lint* analyzes your C programs (one or many modules) and uncovers glitches, bugs, quirks, and inconsistencies. It will catch subtle errors before they catch you. PC–Lint resembles the Lint that runs on the UNIX* O.S., but with more features and some awareness of the 8086 environment.

- Full K&R C
- Supports Multiple Modules–finds inconsistencies between declarations and use of functions and data across a set of modules comprising a program.
- Compares function arguments with the associated parameters and complains if there is a mismatch or too many or too few arguments.
- User-modifiable library description files for most major compilers.
- All warning and information messages may be turned on and off globally or locally (via command line and comments) so that messages can be tailored to your programming style.
- All command line information can be furnished indirectly via file(s) to automate testing.
- Use it to check existing programs, programs about to be exported or imported, as a preliminary to compilation, or prior to scaling up to a larger memory model.
- All one pass with an integrated preprocessor so it's very fast.
- Has numerous flags to support a wide variety of C's, memory models, and programming styles.
- **Price: $139.00 MC, VISA**
  (Includes shipping and handling) PA residents add 6% sales tax. Outside USA add $10.00
- Runs under MS-DOS* 2.0 and up, with a minimum of 128Kb of memory. It will use all the memory available.

Trademarks: PC–Lint (Gimpel Software), UNIX AT&T), MS-DOS (Microsoft)).

NEW!! Amiga - Lint
Special Introductory Price
$98.00

GIMPEL SOFTWARE
3207 Hogarth Lane • Collegeville, PA 19426
(215) 584-4261

Circle Number **11** on the Reader Service Card

It Is My Firm's Resolution to Have More Resolution

Startup Azuray Inc. of Scotts Valley (408-438-7460) has a proprietary CRT technology that produces photographic quality graphics. They have a noninterlaced monitor with 2048 by 2048 resolution with 8 to 12 bits per pixel (256 to 4,096 monochrome shades). You may say that that's a lot of pixels to update, so it must be slow. Well Azuray solved that problem by having *8* different and parallel electron beams trace the screen. In addition, there is *no* shadow mask; the controllers guide the .003 inch diameter electron beams. Company Vice President Roy Murphy says that by increasing the number of beams, it is relatively easy to develop monitors of even greater resolution! To top that off, since each beam is driven by a TV-like signal, the image of an Azuray monitor can be transmitted to any TV receiver in the world by using n separate television channels, where n is the number of beams in the monitor! Current configurations include graphics controller, video controller, a 68000 controlling the traffic, and the ability to store two high resolution image planes. But don't rush out to get one; they cost $30,000 each!

Siemens Communications Systems of Westlake Village, Maxtor Corporation of San Jose, and Advanced Storage Technology of Milpitas, all of California, are producing 5 1/4 inch Winchester drives that are over 300 Megabytes! Siemens Model 1300 offers 306 megabytes of unformatted storage for $3,900. Siemens also expects to introduce a 500 megabyte unit before Fall Comdex.

A 96 Bit Computer

Now that the Intel 80386 32 bit microprocessor has been announced, Altos Computers is working on a powerful multi-user computer that can service 64 users. The trick is to use *3* of these Intel chips to improve throughput. It should be ready by the middle of 1988 and cost around $5,000 for a bare-bones machine. The OS will be none other than UNIX (most likely, the XENIX 5.0 flavor or whatever version is available then). Expect it to be called the 6486 (a sixty four user machine based on the 80X86 family).

A GEM that Is Not a Win-Dough Program

Digital Research is not the only one with a GEM product. Syte Information Technology, currently defunct, is trying to sell the company's main software technology to recoup some operating losses. It's called GEM, Global Environment Manager. Supposedly, it is an operating system kernel that can connect different operating system environments on different hardware transparently. Apparently Syte could not convince its financers that their program was a diamond in the rough.

Thought of the Month

If software is licensed and *not* purchased and owned outright, why do we (the consumer) have to pay sales tax? □

Readers may write to Frank D. Greo

c/o Programmer's Journal
P.O. Box 30160
Eugene, OR 97403

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Tools for the Programmer from Blaise Computing

Save Up To $130 On These Special Offers!

TOOLS & TOOLS 2

For C or Pascal

For a limited time, pick up both packages and save $50 off our regular list price. The C version comes with libraries for the Lattice, Computer Innovations and Microsoft (version 2.03 and 3.00) compilers. The Pascal version supports IBM and Microsoft Pascal. **$175.**

VIEW MANAGER With Source

All libraries are included. Please specify C or Pascal. Regular $425. Save $130. **$295**

Blaise Computing provides a broad range of fine programming tools for Pascal and C programmers, with libraries designed and engineered for the serious software developer. You get clearly written code that's fully commented so that it can serve both as a model and also be easily modified to grow with your changing needs. Our packages are shipped to you complete with comprehensive manuals, sample programs and source code.

None of the programs are copy-protected.

FOR C AND PASCAL PROGRAMMERS:

TOOLS ◇ $125

Extensive string and screen handling, graphics interface and easy creation of program interfaces. Includes all source code.

TOOLS 2 ◇ $100

Memory management, general program control and DOS file support. Interrupt service routine support. Includes all source code.

VIEW MANAGER ◇ $275

General screen management. Create data entry screens that can be easily manipulated from your application program. Block mode data entry and retrieval with fast screen access.

VIEW LIBRARY Source ◇ $150

Source code to the VIEW MANAGER library functions.

ASYNCH MANAGER ◇ $175

Powerful asynchronous communications library providing interrupt driven support for the COM ports. All source code included.

FOR THE TURBO PASCAL PROGRAMMER:

Turbo POWER TOOLS ◇ $99.95

Extensive string support, extended screen and window management; interrupt service routines; program control and memory management; interrupt filters. All source code included.

Turbo ASYNCH ◇ $99.95

Interrupt driven asynchronous communication support callable from Turbo Pascal. ASYNCH is written in assembler and Turbo Pascal with all source code included.

PACKAGES FOR ALL PROGRAMMERS:

EXEC ◇ $95

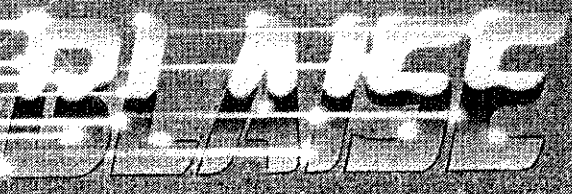
Program chaining executive. Chain one program from another even if the programs are in different languages. Common data area can be specified. Source code included if you're a registered C TOOLS and C TOOLS 2 user.

SPARKY ◇ $75

Run-time resident (or stand-alone) scientific, fully programmable, reverse polish notation calculator. No limit on stack size, variables or tape. Includes all standard scientific functions and different base arithmetic.

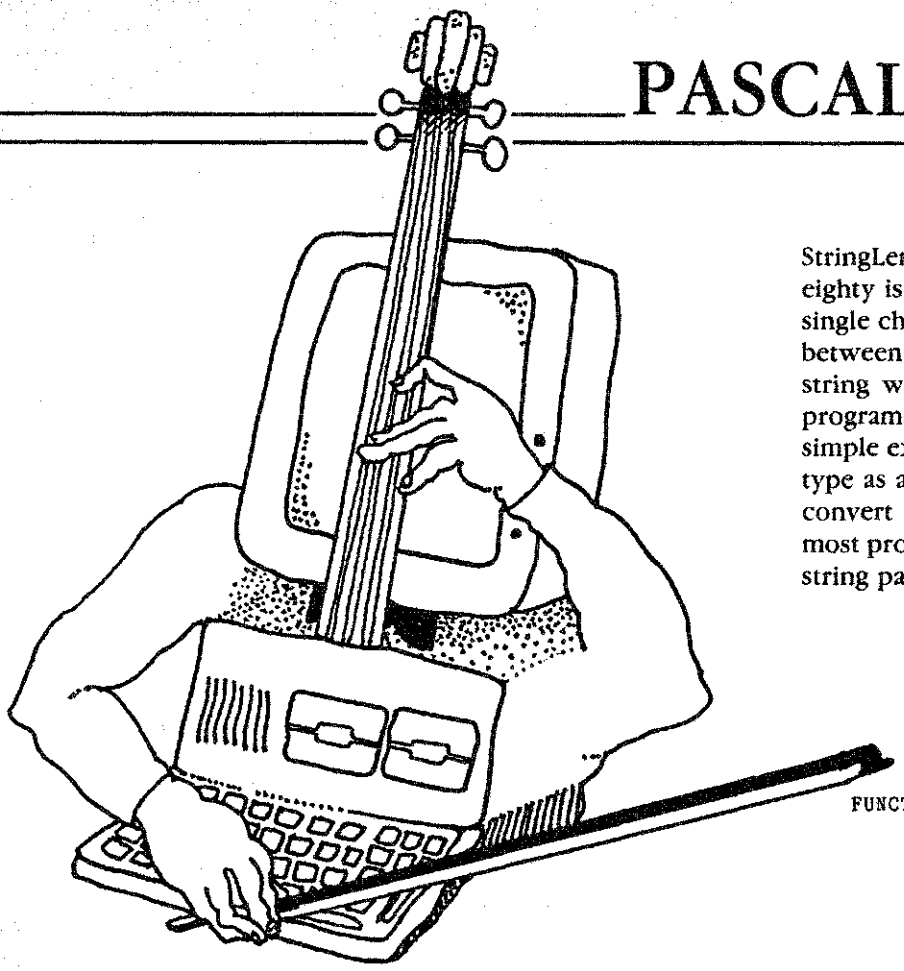
TO ORDER, call Blaise Computing Inc. at (415) 540-5441

◆ 2034 Blake Street ◆ Berkeley, CA 94704 ◆ (415) 540-5441 ◆



Circle Number 2 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



FUNKY STRING FUNCTIONS

by Tom Swan

I've always thought string functions in Pascal would be funky. That's "funky" in its jazz sense—something basic in good music; something sexy. After all, most modern Pascal compilers come with built-in string functions, and therefore, the compiled code obviously has the ability to correctly handle strings as function values. Why, then, do those same compilers prevent you from writing your own string functions? There seems to be no logical answer. Apparently, Borland International came to the same conclusion when designing Turbo Pascal, one of the few Pascal compilers that allow you to write your own string functions. With that ability, as I'll explain here, you can build a useful library of string handling routines with a minimum of programming statements.

Creating a String Function

All Pascal function and parameter data types must be simple identifiers. You cannot declare a parameter as a complex record without first defining the record structure in a TYPE declaration. The same rule applies to string function results. Therefore, the first step in writing a string function is to declare a new data type:

```
CONST
   StringLength = 80;
   Blank = ' ';
TYPE
   OneString = String[ StringLength ];
```

StringLength can be any value from one to 255 although eighty is probably adequate for most programs. Blank is a single character—be sure not to type more than one space between the quotes. OneString is the new data type, a string with the maximum length shown in brackets. All programs in this article require the above declarations. A simple example shows how to use the new OneString data type as a function result in a common string operation to convert all characters in a string to uppercase. Although most programmers code this as a procedure with a variable string parameter, it makes a useful function.

```
FUNCTION UpString( s : OneString ) : OneString;
        { Convert s to all uppercase }
        VAR
           i : 1 .. StringLength;
        BEGIN
           UpString[0] := s[0];  { Length }
           FOR i := 1 TO Length( s ) DO
              UpString[i] := UpCase( s[i] )
        END; { UpString }
```

There are several important features in this example that may not be obvious. First, parameter s is passed by value. Don't be tempted to change this to a VAR parameter, which has the effect of passing the string's address and not the string itself. As written, the value parameter allows passing the results of other string functions as parameters to this and other functions, a fact that will become more important later when we examine string functions that call others in the library. The string function result is, of course, the new OneString data type. You can use the result anywhere a string of the same type could normally be used. To display the uppercase conversion of a string variable FileName, for example, you could write

```
Writeln( UpString( FileName ), ' not found.' );
```

One thing you cannot do, however, is index the result of a string function. You can index only string variables. For that reason, the following statement does not make it through the compiler.

```
Write( UpString(s)[4] );
```

To index a string function result, assign it to a string variable and then index like this:

```
s := UpString( s );
Write( s[4] );
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

That example brings up another interesting point. The function does not change the value of its passed parameter. It returns a new string, in this example converted to uppercase, but makes no alterations on its parameter. To change the parameter, reassign the function result as shown above. Knowing this, you can directly compare two mixed-case strings without affecting either one.

```
IF UpString( A ) < UpString( B )
   THEN SwapString( A, B );
```

Referring back to the UpString function, consider the statement, UpString[0] := s[0]. The purpose of this is to assign the length byte of the original string (stored at s[0]) to the function result. When writing your own string functions, remember to set the result length either as shown in this example or by assigning a string value to the function identifier. Forgetting this important step can lead to serious program bugs, a danger the compiler is not able to catch.

Expanding Your String Library

Another addition to your string function library is shown below. I find new uses for this function over and over.

```
FUNCTION Remove( pattern, s : OneString ) : OneString;
{ Remove pattern (if found) from string }
{ Ignores string case }
VAR p : 0 .. StringLength;
BEGIN
   p := Pos( UpString(pattern), UpString(s) );
   IF p > 0
      THEN Delete( s, p, length( pattern ) );
   Remove := s
END; { Remove }
```

Remove searches for a pattern string in parameter s, and if it finds that pattern, takes it out of the string. This version ignores case in both the pattern and the original string by converting pattern and s to uppercase with the previously described UpString function. If you want to restrict patterns to exact upper- and lowercase matches, change the first statement to read:

```
p := Pos( pattern, s );
```

Remove comes in handy for stripping phrases or characters from strings. You can use it to chop off file name extensions. For example:

```
Remove( '.PAS', FileName );
```

removes the extension .PAS from FileName if found. Another use for Remove is to delete certain characters from strings. This takes out all the blanks in string Original.

```
REPEAT
   OldLen := Length( Original );
   Original := Remove( Blank, Original )
UNTIL Length( Original ) = OldLen;
```

The integer variable OldLen controls the REPEAT . . . UNTIL loop, causing it to end when Remove finds no more blanks in Original.

Stripping from Head to Tail

Stripping the blanks from the heads and tails of strings is probably best done by writing two separate functions to take off the leading and trailing blanks. First we need DeHead:

```
FUNCTION DeHead( s : OneString ) : OneString;
{ Remove blanks from head of string }
BEGIN
   WHILE ( s[1] = Blank ) AND ( Length(s) > 0 ) DO
      Delete( s, 1, 1 );
   DeHead := s
END; { DeHead }
```

And then we need DeTail:

```
FUNCTION DeTail( s : OneString ) : OneString;
{ Remove blanks from tail of string }
VAR
   NewLen : integer;
   FoundChar : Boolean;
BEGIN
   NewLen := Length( s );
   FoundChar := False;
   WHILE ( NewLen > 0 ) AND ( NOT FoundChar ) DO
   BEGIN
      FoundChar := ( s[ NewLen ] <> Blank );
      IF NOT FoundChar
         THEN NewLen := NewLen - 1
   END; { while }
   DeTail := Copy( s, 1, NewLen )
END; { DeTail }
```

DeHead is easy to follow. It simply removes the leading blanks from its parameter, s. The WHILE loop may appear incorrect on close examination. As written, a range error occurs on comparing s[1] = Blank when the string length is zero. However, because range checking is normally off in Turbo Pascal, the error does not cause a program failure, and in fact, the WHILE loop operates correctly. DeTail may be more difficult to understand. The WHILE loop sets NewLen equal to the position of the last non-blank character in s. It then copies all characters up to that position in the original to the function result. Now that we have DeHead and DeTail, it's easy to write a function to remove both leading and trailing blanks from a string.

```
FUNCTION Strip( s : OneString ) : OneString;
{ Remove head and tail blanks from string }
BEGIN
   Strip := DeHead( DeTail( s ) )
END; { Strip }
```

As you can see, string functions often lead to one-statement solutions to problems that would otherwise take several statements to solve. (For an example, see my article "Scanning in Turbo Pascal" in *PJ*, Vol. 3, No. 2.) Also, because DeHead and DeTail are separately debugged, you can be confident that Strip will work as planned. Building new functions and procedures from a debugged library this way can save a lot of programming time.

Strings and Tokens

I define a token as an element of a closed universe. By that definition, the words in a sentence are tokens of English (or another) language. Tokens in programming usually refer to various commands, symbols, and other programming elements. Programs often need to separate the tokens from an input string. For example, you might want someone to

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Fast Path for New Users of Turbo Pascal...

For TURBO PASCAL On The IBM PC

TRAINING WHEELS™

- A comprehensive step-by-step example to help you get started.
- 30 useful & necessary procedures—ready to be included in your program—each with source code and with a simple program illustrating its use: Display help screens & menus quickly, check the DOS environment strings, determine where the program was loaded from, manipulate the archive & read-only bits, determine the ROM date & type of machine & type of display being used, retrieve and parse the command line arguments, make your program preserve the color attributes set for DOS, move a file to a different directory without copying it—and many more.
- Helps get your application going quicker and provides examples of how the DOS function calls are used from Turbo.
- 80-page user-guide plus diskette.

Available for only $25.00
(plus $2 shipping). Cal. Res. add 7%.
Telephone Orders Welcomed (408) 733-4032

LANDMARK SOFTWARE
1142 Pomegranate Court
Sunnyvale, CA 94087

Circle Number 12 on the Reader Service Card

enter a list of commands and then have the program separate and act on each command one at a time. If we define a word as a group of characters not broken by blanks, then the following string function removes the first word from a string:

```
Word( s : OneString ) : OneString;
        { Return next word from string }
        VAR
           p : 0 .. StringLength;
        BEGIN
           s := Strip( s );
           p := Pos( Blank, s );
           IF p = 0 THEN p := Length( s );
           Word := Strip( Copy( s, 1, p ) )
        END; { Word }
```

Word works by first stripping all blanks from parameter s and then setting variable p to the position of the first blank found in the result. If p is zero, then Word should return the entire string—it contains only a single token. Otherwise, Word returns the characters up to the first blank by copying characters from s[1] to s[p]. Notice how the final statement strips extra blanks from the function result. Of course, Word returns only the first token. To get to the others, we have to remove the result of Word from the original string. In this case, a procedure instead of a function seems best.

```
PROCEDURE Token( VAR Source, NextToken : OneString );
BEGIN
   NextToken := UpString( Word( Source ) );
   Source := Strip( Remove( NextToken, Source ) )
END; { NextToken }
```

This two-statement function employs all of the preceding string functions, demonstrating how complex programs can often be constructed from a well-designed library of general purpose routines. Token works by setting variable parameter NextToken to the first Word, converted to all uppercase, in parameter Source. It then Removes that token from Source, setting Source to the Stripped result. By coding the procedure into a simple loop, as the following example demonstrates, you can separate the words from any string. (The example requires all previous function, procedure, and global declarations.)

```
TestToken;
        VAR
           Original, NextToken : OneString;
           Done : Boolean;
        BEGIN
           Writeln( 'Test Token procedure' );
           REPEAT
              Writeln;
              Writeln( 'Enter a string: ' );
              Readln( Original );
              Done := ( Length( Original ) = 0 );
              WHILE Length( Original ) > 0 DO
              BEGIN
                 Token( Original, NextToken );
                 Writeln( NextToken )
              END { while }
           UNTIL Done
        END.
```

Function for Number Formatting

Pascal's numeric formatting in Writeln statements is simplistic. You can specify the number of columns (plus the number of decimals for real numbers), but that's it. Even the simple job of displaying leading zeros, handled by most BASIC interpreters with a PRINT USING command, takes special programming in Pascal. With string functions, you can add more sophisticated numeric formatting to Pascal. To add leading zeros to integer values, you could write a string function such as this:

```
FUNCTION Number( n : integer; d : integer ) : OneString;
{ Convert n to a (d+1)-column string with leading zeros
VAR
   s : OneString;
   ch : char;
BEGIN
   Str( abs( n ), s );
   WHILE Length( s ) < d DO
      Insert( '0', s, 1 );
   IF n < 0 THEN ch := '-' ELSE ch := Blank;
   Number := ch + s
END; { Number }
```

The built-in Str function converts n to a string. Although you could write Str(n:d,s) to right justify n in d columns, that still doesn't add the leading zeros we want. Also, the absolute value of n is used to place the minus sign to the left of the first leading zero. (Change the statement to Str(n,s) and enter a negative number to see why this is necessary.) After inserting up to d leading zeros, the minus sign is properly attached to the string function result. A simple test program shows how to use the function. (Before compiling, add the global declarations from the beginning of this article along with the Number function.)

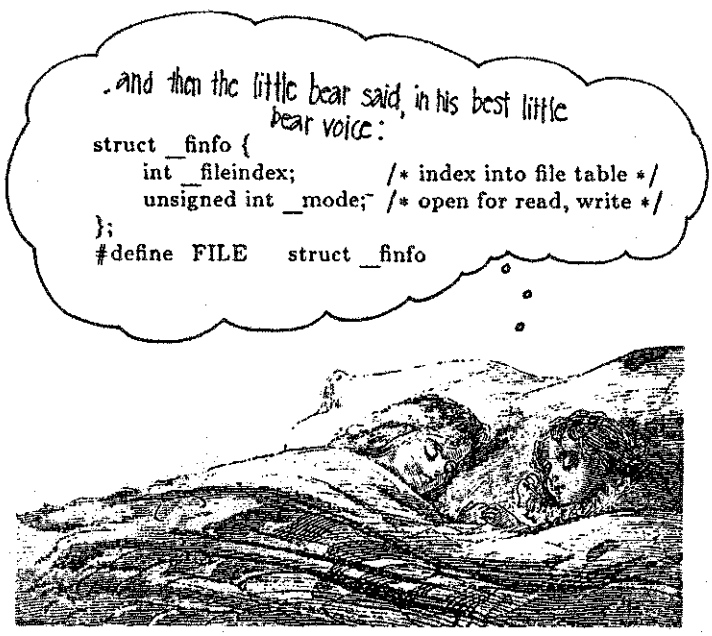
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
PROGRAM TestNumber;
VAR
   n, Digits : integer;
BEGIN
   Writeln( 'Test Token procedure' );
   REPEAT
      Writeln;
      Writeln( 'Enter an integer (-1 to quit): ' );
      Readln( n );
      FOR Digits := 1 TO 8 DO
         Write( Number( n, Digits ), Blank:2 )
   UNTIL n = -1
END.
```

Disadvantages of String Functions

There are a few disadvantages that come with the string functions described in this article. For instance, because parameters are passed by value, calls to functions must first copy entire string parameters onto the stack. Programs usually run faster when complex structures are passed by reference as VAR parameters instead. Along the same lines, value parameters take more stack memory than equivalent VAR parameters. In cases such as the Token procedure, cascaded function calls place many copies of their parameters on the stack. This can lead to the inefficient use of memory. Another disadvantage is that programs may be difficult to convert to other Pascals that do not allow string functions. Despite these disadvantages, minor for my purposes, I was glad to see string functions in Turbo Pascal. They may not play at parties, but they're funky music to me. □

Tom Swan is the author of several books on Pascal programming including Pascal Programs for Business, Pascal Programs for Games and Graphics, Pascal Programs for Data Base Management, *and most recently,* Mastering Turbo Pascal *published by Hayden Book Company. Address correspondence to Swan Software, PO. Box 206, Lititz, PA 17543. Please enclose a self-addressed, stamped envelope if you want a personal reply.*


...and then the little bear said, in his best little bear voice:
struct _finfo {
int _fileindex; /* index into file table */
unsigned int _mode; /* open for read, write */
};
#define FILE struct _finfo

PRODUCTIVITY UTILITIES

From Opt – Tech Data Processing

NEW **On-Line Help** NEW

A comprehensive utility for adding help windows to your programs. It provides an efficient method for storing help texts in libraries and standard routines for interfacing your programs with these libraries.

Help windows are displayed in a fraction of a second. You have total control over the contents of the window, its position on the screen and the display colors used.

On-Line Help can be interfaced with interpreted Basic and all popular compilers.

On-Line Help includes a Help Library Manager, Library Interface Routines and source code examples. **$149.**

Opt-Tech Sort

High performance sort/merge utility can be used as a stand-alone program or called as a subroutine to many languages.

Supports unlimited filesizes, multiple input files and fixed or variable length records. Many special file types are supported including Btrieve and Dbase. Up to nine sort control fields (ascending or descending), all common data types supported. Output file can be full records, keys or pointers.

Written in assembly language for high performance. Example: 4,000 128 byte records sorted to give key and pointer in 30 seconds. **$99.**

Scroll & Recall

Allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 screens of data can be recalled or written to a disk file (great for documenting systems operations).

Allows you to easily recall and edit your previously entered DOS commands and data lines without re-typing.

Scroll & Recall is very easy to use. It's a resident utility that's always there when you need it! **$69.**

All programs are fully documented and are compatible with the IBM-PC, XT, AT and compatible machines using the DOS operating system.

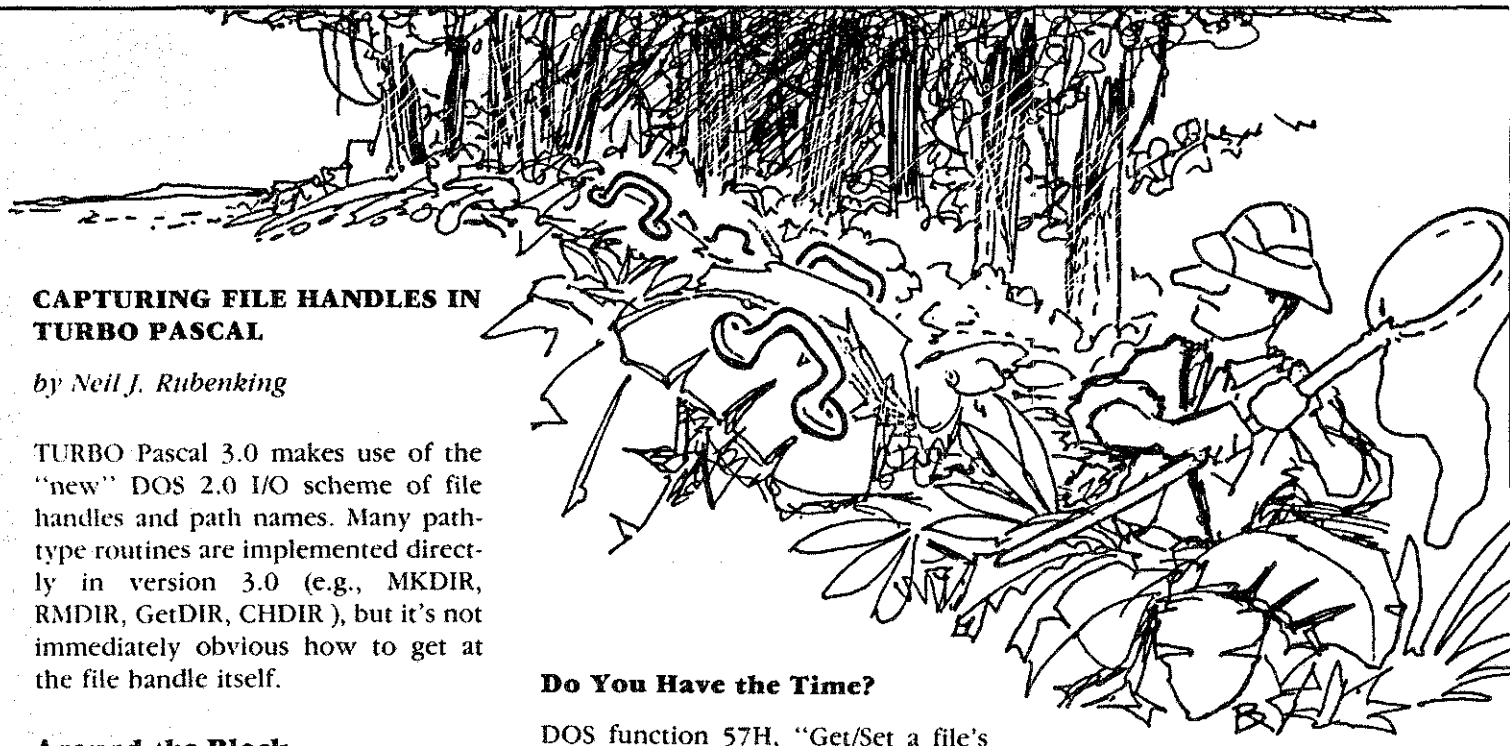
Visa, M/C, Check, Money Order,
COD, or PO accepted.
Quantity and Dealer discounts available.

To order or to receive additional information write or call:

Opt-Tech Data Processing
P.O. Box 678 — Zephyr Cove, NV 89448
(702) 588-3737

Circle Number **16** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



CAPTURING FILE HANDLES IN TURBO PASCAL

by Neil J. Rubenking

TURBO Pascal 3.0 makes use of the "new" DOS 2.0 I/O scheme of file handles and path names. Many path-type routines are implemented directly in version 3.0 (e.g., MKDIR, RMDIR, GetDIR, CHDIR), but it's not immediately obvious how to get at the file handle itself.

Around the Block

Any TURBO file variable has an associated File Interface Block. The exact structure of the block differs for text and non-text files, but in both cases the first two bytes in the block are the handle and the 13th through 76th contain the pathname associated with that file as an ASCIIZ string (a string of characters ended by an ASCII zero character). This structure is matched in the sample program by the record type "FIB"—specifically, this is the structure for a non-text file. The text-file FIB structure is included as "tFIB", but is not used.

To get access to the FIB for a given file, we just place a variable of type FIB "on top of" the file variable with an absolute declaration. In order to make this routine work for any file, we have to avoid declaring a variable parameter of a particular file type. That is possible with TURBO's "untyped variable parameter"—the parameter is declared VAR in the procedure block, but no type is given. Access to the parameter is through its address, usually by declaring a variable *absolute* at that address. See the procedures "HandleOf" and "PathOf" in the sample program.

Do You Have the Time?

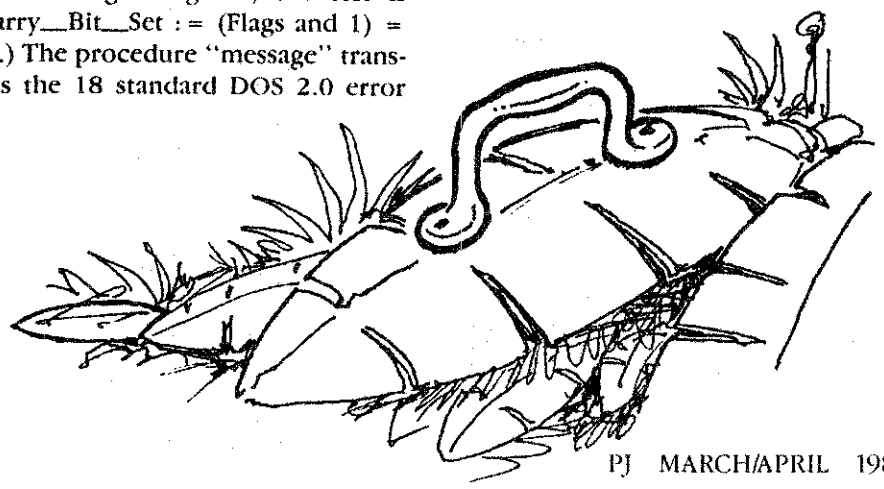
DOS function 57H, "Get/Set a file's date and time," requires the file handle. The procedure "DateAndTime" in this sample program uses TURBO's Intr procedure to invoke function 57H and returns the date and time coded as two integers. Functions "TransDate" and "TransTime" translate these two codes into a legible form.

Error Massage

It's possible that even a simple item like the date/time function could fail, so we have to be prepared for an error message. Most of the DOS functions that can be called with the "MSDOS" routine set the carry flag if an error occurs and leave an error code in AX. (Since the carry flag is the lowest bit in the "Flags" register, the test is "Carry_Bit_Set := (Flags and 1) = 1;".) The procedure "message" translates the 18 standard DOS 2.0 error messages into words—it can be used to interpret errors for other DOS functions, too.

Got a Handle on It?

Your programs can use "HandleOf" on any *open* file—closed files will return a handle of $FFFF. "PathOf" can be used in debugging to return the current filename associated with a file variable, or perhaps a RAM-resident program might want to find out the names of the files associated with all handles. "DateAndTime" can SET the date and time of a file, too, as long as you write inverse functions to TransDate and TransTime. ('But Boss, look at the date stamp on these files—I was working all weekend!') □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

①

```
PROGRAM grab_handle;

    (*******************************************************)
    (*  copyright 1985 by Neil J. Rubenking                *)
    (*     Requires TURBO 3.00, PC-DOS 2.xx+               *)
    (*                                                     *)
    (*******************************************************)
    (*       This program demonstrates getting a file's    *)
    (*  handle and using it to get the date and time stamp *)
    (*  for that file.                                     *)
    (*******************************************************)

  TYPE

    FIB = RECORD
            Handle   : Integer;
            RecLen   : Integer;
            UnUsed   : ARRAY[1..8] OF Byte;
            FilePath : ARRAY[1..64] OF Char;
          END;

    tFIB = RECORD
             Handle      : Integer; {-----------------}
             flags       : Byte;    { This is the FIB }
             Ch_Buff     : Char;    { structure for   }
             Buff_Offset : Integer; { a TEXT file.    }
             Buff_Size   : Integer; {-----------------}
             Buff_Pointer : Integer;
             Buff_End    : Integer;
             FilePath    : ARRAY[1..64] OF Char;
           END;

    regtype = RECORD
                ax, bx, cx, dx, bp, si, di, ds, es, flags : Integer;
              END;

    filename = STRING[64];
    string40 = STRING[40];
    string9  = STRING[9];
    string4  = STRING[4];

  VAR                                  {-----------------}
    F             : Text;              { Three different }
    G             : FILE OF string4;   { kinds of files. }
    H             : FILE;              {-----------------}
    Handle, code  : Integer;
    Regs          : regtype;
    name          : filename;
    errr          : Byte;
    dat, tim      : Integer;
```

②

```
  (*******************************************************)
  (*  The function MESSAGE can be used after any call    *)
  (*  to the MSDOS or INTR procedures that returns an    *)
  (*  error message in AX.  Usually the signal that AX   *)
  (*  contains an error message is the CARRY FLAG being  *)
  (*  set to 1.  This is tested equivalent to the state- *)
  (*  ment "flags and 1 = 1."                            *)
  (*******************************************************)

FUNCTION message(code : Byte) : string40;
  BEGIN
    CASE code OF
      1  : message := 'Invalid function number.';
      2  : message := 'File not found.';
      3  : message := 'Path not found.';
      4  : message := 'Too many open files (no handles left).';
      5  : message := 'Access denied.';
      6  : message := 'Invalid handle.';
      7  : message := 'Memory control blocks destroyed';
      8  : message := 'Insufficient memory';
      9  : message := 'Invalid memory block address.';
      10 : message := 'Invalid environment.';
      11 : message := 'Invalid format.';
      12 : message := 'Invalid access code.';
      13 : message := 'Invalid data.';
      15 : message := 'Invalid drive was specified.';
      16 : message := 'Attempt to remove current directory.';
      17 : message := 'Not same device.';
      18 : message := 'No more files.';
    ELSE message := 'Error.'
    END;
  END;



  (*******************************************************)
  (*  The DOS function $57 takes the handle of an open   *)
  (*  file as input (in BX).  It returns that file's     *)
  (*  date and time coded as integers in DX and CX.  If  *)
  (*  the operation is not successful, the Carry Flag    *)
  (*  is set and an error code is returned in AX.  If    *)
  (*  happens, the program interprets the error code and *)
  (*  halts.                                             *)
  (*******************************************************)

PROCEDURE DateAndTime(han : Integer; VAR DT, TM : Integer);
  BEGIN
    WITH Regs DO
      BEGIN
        ax := $57 SHL 8;
        bx := han;
        MsDos(Regs);
        IF flags AND 1 = 0 THEN
          BEGIN
            DT := dx;
            TM := cx;
          END
        ELSE
          BEGIN
            WriteLn(#7, message(ax));
            Halt;
          END;
      END;
  END;
```

③

```
  (*******************************************************)
  (*  To get a file's handle, we simply define an FIB    *)
  (*  variable "on top of" the file variable.  Making    *)
  (*  the input variable an UNTYPED VAR PARAMETER ensures*)
  (*  that we can use this routine with any type of file.*)
  (*  Since the handle is the first integer in the FIB   *)
  (*  for both text and non-text, this function is valid *)
  (*  for any (open) file.                               *)
  (*******************************************************)

FUNCTION HandleOf(VAR marker) : Integer;
  VAR
    temp : FIB ABSOLUTE marker;
  BEGIN
    HandleOf := temp.handle;
  END;

  (*******************************************************)
  (*  Since your program opened the file, you almost     *)
  (*  certainly KNOW what the filename was.  But just    *)
  (*  to show it can be done, here we query the FIB as   *)
  (*  to what filename was used to open it.  Again, we   *)
  (*  just define a FIB variable on top of the file      *)
  (*  variable and access it directly.                   *)
  (*      The pathname is stored as an ASCIIZ string --  *)
  (*  a string of characters terminated by a chr(0).  We *)
  (*  translate it into a normal TURBO string.           *)
  (*******************************************************)

FUNCTION pathOf(VAR marker) : filename;
  VAR
    tempFIB : FIB ABSOLUTE marker;
    TmpName : filename;
    N       : Byte;
  BEGIN
    N := 1;
    TmpName := '';
    REPEAT
      TmpName := TmpName+tempFIB.FilePath[N];
      N := N+1;
    UNTIL tempFIB.FilePath[N] = #0;
    pathOf := TmpName;
  END;

  (*******************************************************)
  (*  The "exists" function uses TURBO's ability to shut *)
  (*  of I/O checking to determine whether a file exists.*)
  (*  If it does not, trying to reset it will produce an *)
  (*  error code in the pre-defined variable "IOResult". *)
  (*  This same technique can be used with "ChDir" to    *)
  (*  determine whether a sub-directory is valid.        *)
  (*******************************************************)

FUNCTION exists(fn : filename) : Boolean;
  VAR
    temp : FILE;
  BEGIN
    Assign(temp, fn);
    {$I-}
    Reset(temp);
    {$I+}
    exists := IOResult = 0;
    Close(temp);
  END;
```

④

```
  (*******************************************************)
  (*  This function has a single purpose -- it translates*)
  (*  an integer-encoded DOS date into a string of the   *)
  (*  form "07-20-53".                                   *)
  (*                                                     *)
  (*  The date is encoded in 16 bits as follows:         *)
  (*      yyyyyyymmmmddddd                               *)
  (*                                                     *)
  (*  To get at one of the portions of the date, first   *)
  (*  "mask off" the extraneous bits.  To get the month, *)
  (*  for example, we want only bits 5 thru 8.  If we    *)
  (*                       AND the binary number at left *)
  (*  yyyyyyymmmmddddd     with the date, all bits but the*)
  (*  0000000111100000     four MONTH bits will be set to*)
  (*                       zero.  To read the month value*)
  (*  we need only shift it to the right by 5 binary     *)
  (*  digits.                                            *)
  (*******************************************************)

FUNCTION DateF(D : Integer) : string9;
  VAR
    day, mo, yr : Byte;
    temp        : string9;
    portion     : STRING[2];
  BEGIN
    yr  := (D AND $FE00) SHR 9;
    mo  := (D AND $1E0) SHR 5;
    day := (D AND $1F);
    Str(mo:2, portion);
    temp := portion+'-';
    Str(day:2, portion);
    temp := temp+portion+'-';
    {---------------------------}
    { Year values 0 to 119 stand  }
    { for 1980 to 2099.           }
    {---------------------------}
    yr   := (yr+80) MOD 100;
    Str(yr:2, portion);
    temp := temp+portion;
    IF temp[4] = ' ' THEN temp[4] := '0';
    IF temp[7] = ' ' THEN temp[7] := '0';
    DateF := temp;
  END;

  (*******************************************************)
  (*  The TIME conversion is effectively the same as the *)
  (*  DATE.  The encoding is slightly different:         *)
  (*      hhhhhmmmmmmssssss                              *)
  (*                                                     *)
  (*  The "seconds" are actually two-second units.  DOS  *)
  (*  does not display the seconds when you ask for a    *)
  (*  directory, but the data is available.  The decoded *)
  (*  time is here displayed in the form "11:30.20a"     *)
  (*******************************************************)

FUNCTION TimeF(T : Integer) : string9;
  VAR
    hours, mins, secs : Byte;
    temp              : string9;
    portion           : STRING[2];
    a_or_p            : Char;
  BEGIN
    hours := (T AND $F800) SHR 11;
    mins  := (T AND $7E0) SHR 5;
    secs  := (T AND $1F)*2;
    CASE hours OF
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
  0, 24: BEGIN
           hours  := 12;
             a_or_p := 'a';
           END;
         1..11 : a_or_p := 'a';
           12 : a_or_p := 'p';
         13..23 : BEGIN
                    hours := hours-12;
                      a_or_p := 'p';
                    END;
       END;
       Str(hours:2, portion);
       temp := portion+':';
       Str(mins:2, portion);
       temp := temp+portion+':';
       Str(secs:2, portion);
       temp := temp+portion+a_or_p;
       IF temp[4] = ' ' THEN temp[4] := '0';
       IF temp[7] = ' ' THEN temp[7] := '0';
       TimeF := temp;
     END;

BEGIN
  ClrScr;
  dat := 0; tim := 0;
  writeln('                                    TURBO FIB DEMONSTRATION');
  writeln;
  Write('Enter the name of an existing file: ');
  ReadLn(name);
  IF exists(name) THEN
    BEGIN
      Assign(F, name);
      Assign(G, name);
      Assign(H, name);
      Reset(F);
      Reset(G);
      Reset(H);
      WriteLn;
      Write('  Three files have been assigned to ', name);
      WriteLn(' File "F" is a text file,');
      Write('  "G" is a file of records, and "H" is an ');
      WriteLn('untyped file.  The FIB routines');
      WriteLn('  work equally well on all three.');
      WriteLn;
      WriteLn('Handle of F IS ', HandleOf(F));
      WriteLn('Handle of G IS ', HandleOf(G));
      WriteLn('Handle of H IS ', HandleOf(H));
      WriteLn;
      WriteLn('F was opened as "', pathOf(F), '"');
      WriteLn;
      DateAndTime(HandleOf(F), dat, tim);
      Close(F);
      Close(G);
      Close(H);
      IF dat+tim <> 0 THEN
        BEGIN
          Write('File "', name, '" was created on ');
          WriteLn(DateF(dat), ' at ', TimeF(tim));
        END
      ELSE WriteLn('ERROR');
    END
  ELSE
    WriteLn('ERROR -- File "', name, '" does not exist!');
END.
```


WINDOWS GIVES YOU — LESS . . . . . *FOR*. . . . . MORE!

It's true. **WINDOWS** gives you less code and shorter development cycles for more sophisticated, eye-grabbing displays. Suddenly, pop-up windows, pull-down menus, and true context sensitive on-line help functions are a snap. You can drag windows around the screen and display different files in any number of windows. There's no snow and no delay.

WINDOWS is written entirely in assembly language so it's ultra-fast and VERY small: less than 2k (3k for large model). It runs on the IBM PC/XT/AT and compatibles. Being one object module, it's VERY EASY to learn and use.

The chapter on asthetics helps you pick the best color combinations for YOUR APPLICATIONS (hint: go for high contrast and avoid chromostereopsis). There's tricks for using placement, shape and color to maximize the communication quality of your programs.

WINGS LET'S YOU *FLY* THROUGH WINDOWS

For the ultimate in development environments, add ***WINGS*** to your **WINDOWS.** Our Window Generation System lets you create and modify windows in only seconds. A single keystroke then writes commented source code for your custom windows. With ***WINGS*** it's incredibly easy to experiment with different screen layouts and color schemes. Bottom line: your applications look much more impressive because you spend time optimizing the display, not writing code and recompiling over and over again.

WINDOWS is easy on your pocketbook — $99.95. ***WINGS*** is even easier — $49.95.

C WINDOWS provides ALL object modules for all memory models of Computer Innovations, Desmet, Lattice, and Microsoft C on the same disk. Support for Aztec and Mark Williams C is coming soon.

BASIC WINDOWS supports Digital Research CB-86 under PC/MSDOS, Microsoft Compiled Basic, Business Basic, and Quick Basic and the IBM Basic Compiler and Business Basic Compiler.

WE'LL INCLUDE FULLY DOCUMENTED SOURCE CODE FOR THE WHOLE **WINDOWS** package — **FREE**

Find out why our windows are in use at AT&T, the Canadian government, Digital Research, Heritage Bank, Lotus, Owens-Corning, Peat Marwick & Mitchell, Security Pacific, Scott Paper, Shell Oil, the U.S. Navy and many major universities . . . **Order today.**

CALL (213) 322-2853

SYSCOM SOFTWARE • 320 W. OAK STREET, EL SEGUNDO, CA 90245

Circle Number **34** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE PROPOSED ANSI C STANDARD

by Rex Jaeschke

© Rex Jaeschke, 1985. All rights reserved.

Welcome to this new series of articles, each of which will discuss one or more aspects of the Proposed ANSI C Language Standard. Emphasis will be placed on the effects of this Standard on MS/PC DOS environments where appropriate, and comments will be made on conforming translators as they become available.

In mid 1983, Jim Brodie of Motorola started the C Language Committee with the sanction of the ANSI X3 Secretariat. The official designation of The Committee is X3J11. Jim is Chairman of The Committee, and other officers include Tom Plum, Vice-Chairman (Plum Hall); P.J. Plauger, Secretary (Whitesmiths, Ltd.); and Larry Rosler, Draft Editor (AT&T).

Some 200 companies and individuals are members of X3J11, with approximately 45 members attending each of the quarterly meetings. The vast majority of attendees at each meeting are implementers, and the mainstream vendors are well represented. Members include AT&T, DEC, Data General, Cray Research, Burroughs, Honeywell, Hewlett Packard, Wang, Harris, Masscomp, Microsoft, Digital Research, Lattice, Computer Innovations, Rational Systems, the *C Journal*, and the */c* magazine.

All members receive a quarterly update of the Draft Standard and copies of official correspondence. Observing members may attend meetings but cannot vote in full committee meetings. Voting members can vote on all issues, but they must maintain a strict meeting attendance record to keep their voting rights current.

The Committee is not a UNIX committee. While the major UNIX and UNIX-like shops are well represented, so too are MS/PC DOS, CP/M (and concurrent DOS), and a host of hardware vendor operating systems. The freestanding implementers (those that run C programs without the benefit of an operating system) are represented by Intel, Zilog and Motorola. In all, the Committee includes a good cross section of C implementers, but sadly, only two or three members actively represent the user community without regard to any particular implementation.

Copies of the Draft Standard are available from

> X3 Secretariat CBEMA
> 311 First Street, N.W.
> Suite 500
> Washington, D.C. 20001
> (202) 737-8888
> Price: $20-$25

For more information about the Committee's activities, contact

> Dr. Thomas Plum
> X3J11 Vice-Chair
> Plum Hall, Inc.
> 1 Spruce Ave
> Cardiff, NJ 08232
> (609) 927-3770

Regular information about the Committee's activities is reported in Tom Plum's Column in

> The C Journal
> InfoPro Systems
> 3108 Route 10
> Denville, NJ 07834
> (201) 989-0570

and

> /c - The Journal for C Users
> Que Publishing, Inc.
> 7999 Knue Road
> Indianapolis, IN 46250

The Committee expects to release a Draft Standard for public comment after the March, 1986, meeting. Comments would then be accepted for one year, with each receiving a formal, written response. After that review period expires, all necessary changes will be made and the Draft will be rereleased for a second review after which time it will be voted on by full Committee (and the X3 Secretariat) before becoming an official ANSI Standard. This whole process may be completed by mid 1987, but as much of the current Draft Standard is unlikely to change, implementers are already beginning to provide some of the new capabilities. Microsoft's V3.0 compiler is one example of this. Until there is an official Standard, however, there can be no conforming implementations.

The Standard includes the preprocessor, the language, and the run-time library, and extensions have been added to each. From this point on, the term C will include all three components of an implementation unless otherwise stated.

Perhaps the single biggest achievement of the Committee has been to decree that a conforming hosted implementation must implement at least all of the Standard library or none of it. A hosted implementation with a subset library does not conform to the Standard. All UNIX specifics have been removed from C unless they could be reasonably implemented in non-UNIX environments, and where existing prior-art (such as exists in UNIX System III and V, 4.XBSD and Whitesmiths' IDRIS system) has shown a reasonable way to go, so too has the Standard often gone.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The aims of the Standard are

1. To preserve the spirit of C as a sparse language and not unduly add to the language: to define and consolidate C, removing all known ambiguities in the definition, particularly in the library.

2. To break as little existing code as possible except where existing practices directly contradict a reasonable, consolidated approach. If code is to be broken, it must be done in such a way that the results are obviously wrong—by way of translation diagnostics and/or obvious run-time errors. (The term "translator" instead of "compiler" will be used throughout as the Committee includes interpreter vendors and the Standard is written to include all current methods of translation.)

3. To add new capabilities that directly add functionality or provide assistance in writing "better" programs. Examples of these are the addition of function prototypes and the variable classes const and volatile.

4. To add capabilities that assist in program portability. Included in this category is the stdarg header, which provides a mechanism for using variable argument lists in a portable fashion.

The Standard contains numerous references to the terms "defined," "implementation-defined," and "undefined." The first indicates that something is guaranteed to behave in a specified way in all conforming implementations. The second means that a behavior depends on a particular hardware and/or software implementation. The third term applies to those situations for which there is no definition. In these cases the results of undefined operations should never be relied on, not even in two different versions of the same translator, not even, perhaps, in multiple occurrences of the same operation within the same program.

Program Startup

The main function seems like a reasonable place to start this report on the proceedings of the X3J11 Committee, and in particular, command-line processing and setting up argc and argv will be discussed first. Throughout this section, only hosted implementations will be considered. Freestanding implementations do not run with the benefit of an operating system, and as such, can be started in any way deemed desirable.

A conforming program may have a main function containing either none or two arguments. The existence and purpose of any extra arguments are implementation-defined. A third argument, envp, is present in UNIX-like environments. Microsoft's V3 compiler includes this third argument to maintain compatibility with its XENIX system. (Actually, their DOS and XENIX compilers are one and the same with minor extensions added to each for operating-specific purposes.) It is recommended that the library function getenv be used instead of the envp argument if portability is a concern.

argv[argc] contains a NULL pointer. This means that the argv array contains (argc + 1) elements, not argc as before. This allows the argv pointer array to be processed without regard to the value of argc.

The Standard makes no comment about the handling of quoted literals in command-lines. Therefore, the ability to handle quoted strings containing whitespace is implementation-defined. If the host environment cannot handle command-line arguments containing both alphabetic cases, it must supply text arguments in lower case.

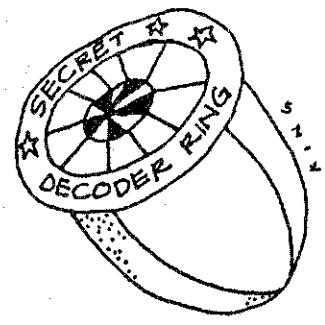
argv[0] represents the program name. If this is not available, *argv[0] must be a '\0'. (Lattice currently supplies the string "c".) In DOS versions 1.X and 2.X, the loader and Program Segment Prefix (PSP) did not store argv[0], so it was not available to C programs. DOS 3.X, however, does store the complete filename and path used to invoke a program. For example: A program TEST.EXE resides in the root directory of Drive B:; if TEST were run under DOS 3.X, argv[0] would be "B:\TEST.EXE".

The parameters argc and argv and the strings pointed to by argv, must be modifiable by the user program, and these may not be changed by the implementation while the user program is executing. Note that a user program includes only that code in and subordinate to main. It does not include any program startup and termination code that is added to the user code to build an executable module as these areas are outside the user program's scope. □

Rex Jaeschke is an independent computer consultant, writer and lecturer. He is a member of the X3J11 Committee and co-founder and Editor of the C Journal. *If you would like to comment on or question any aspect of the Proposed ANSI C Standard, write to Rex at*

2051 Swans Neck Way
Reston, VA 22091

Rex will try to address your concerns in future columns as space permits.

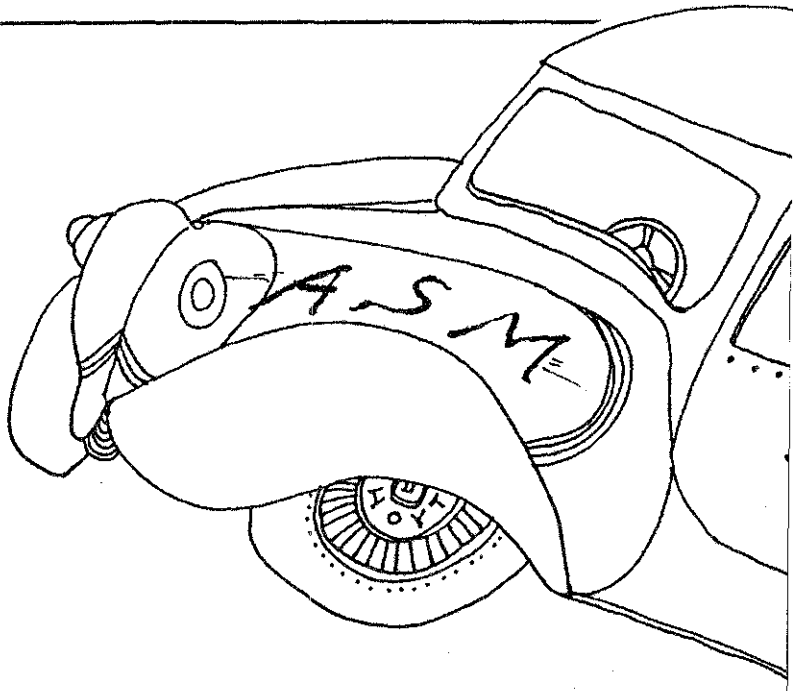


Programmers' Nightmare

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OPTIMIZING FOR SPEED

by Michael Hoyt



One of my areas of interest is graphics, as displayed and as printed. Much of my contract programming is developing specialized screen-dumps from unusual display boards to even more unusual printers.

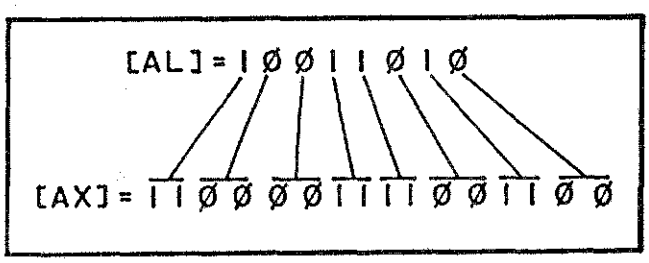
When a printer requires lengthy control sequence strings, it pays to have the program spend as little time as possible twiddling bits before sending them to the printer. Otherwise, the screen-dump takes forever. (Clients DO NOT like to wait for computers. They want instant results.)

This article covers the process I used to optimize a specific conversion routine for speed. I'll not cover the design of the program as a whole or the way it works, other than to say it is very modular in nature.

Conversion

At one (recurring) point in the execution of the program, the [AL] register contains a byte of information corresponding to an 8 pels (1 byte) from the display board's memory. Each bit is set to either 1 or 0, depending on the state of the pel it represents (ON or OFF). For the hard copy produced by the printer to retain the same aspect as the screen, it's necessary to duplicate each bit so that it can be printed twice. This particular printer happens to have a sufficient number of pins, so sixteen (16) can be fired at once.

Since operations performed in the CPU's registers are faster than operations that access memory locations, I decided to design an algorithm that did all its bit manipulation in the CPU. The task of the routine is to take each bit in [AL] and move it the correct number of positions so that [AX] will hold a new value having two (2) bits corresponding to each original bit in [AL]. Here is a graphic representation:



Looks simple enough . . .

Design

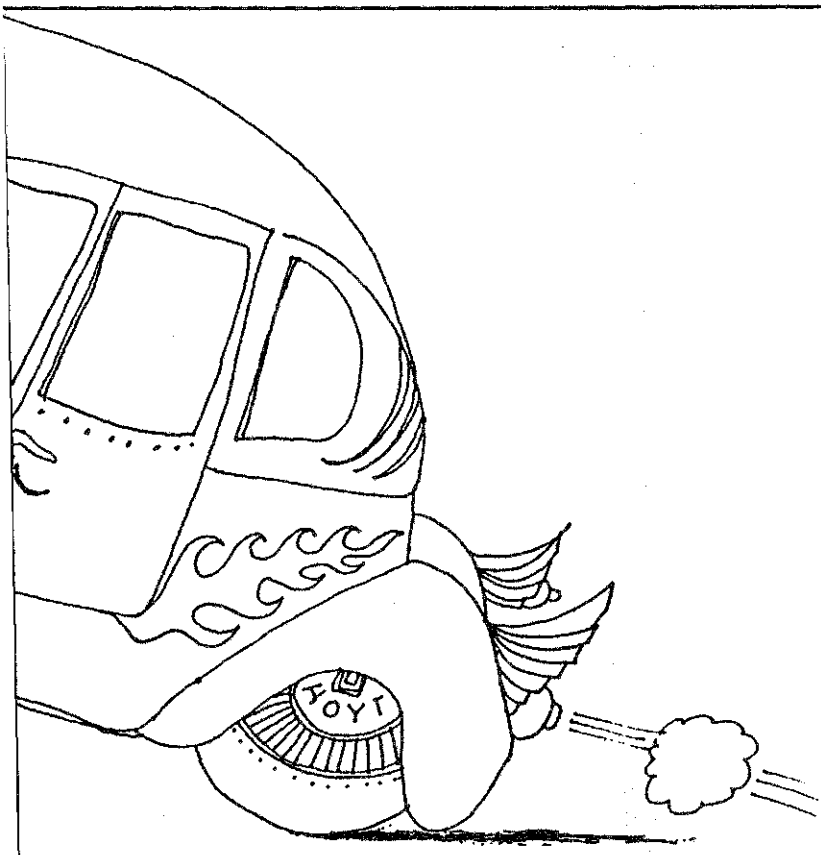
I decided not to use MULtiply or DIVide instructions. Past experience has shown them to be slow, and I wanted speed. Besides, I couldn't readily see a way to efficiently utilize either.

I arrived at a conversion method consisting of three steps. The first involves moving each bit in [AL] to the LSB (least significant bit) of each bit-pair in [AX]. The second step establishes the value of the MSB (most significant bit) of each bit-pair in [AX]. The two (2) values are then combined for the final result.

```
Version 1:

Convert proc near
                                        ;clocks
  push  bx                              ;15
  xor   ah,ah                           ; 3
  mov   bl,al                           ; 2
  and   al,00000001b                    ; 4
Second_Bit:
  test  bl,00000010b                    ; 5
  jz    Third_Bit                       ;16 if jump, 4 if not
  add   ax,0000000000000100b            ; 4
Third_Bit:
  test  bl,00000100b                    ; 5
  jz    Fourth_Bit                      ;16 if jump, 4 if not
  add   ax,0000000000010000b            ; 4
Fourth_Bit:
  test  bl,00001000b                    ; 5
  jz    Fifth_Bit                       ;16 if jump, 4 if not
  add   ax,0000000001000000b            ; 4
Fifth_Bit:
  test  bl,00010000b                    ; 5
  jz    Sixth_Bit                       ;16 if jump, 4 if not
  add   ax,0000000100000000b            ; 4
Sixth_Bit:
  test  bl,00100000b                    ; 5
  jz    Seventh_Bit                     ;16 if jump, 4 if not
  add   ax,0000010000000000b            ; 4
Seventh_Bit:
  test  bl,01000000b                    ; 5
  jz    Eighth_Bit                      ;16 if jump, 4 if not
  add   ax,0001000000000000b            ; 4
Eighth_Bit:
  test  bl,10000000b                    ; 5
  jz    No_More_Bits                    ;16 if jump, 4 if not
  add   ax,0100000000000000b            ; 4
No_More_Bits:
  mov   bx,ax                           ; 2
  shl   ax,1                            ; 2
  add   ax,bx                           ; 3
  pop   bx                              ;12
  ret                                   ;16 (intrasegment)
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



[AL] is used for input to the routine, and output is returned in [AX]. Only one (1) other register is needed by the algorithm [BX], so the number of PUSHes and POPs is limited. It's a good thing because they're not particularly fast.

The routine begins by clearing all but the first bit of [AX] to receive the new values. Since we want to retain the value "as-is," it only adds time to the routine if the value is first cleared then replaced. The original value having been moved from [AL] to [BL], all the bit TESTing is done on [BL]. After each TEST, [AX] is changed only if necessary. Each remaining bit is handled in a similar fashion, and the result of the first step is saved in [BX].

Step two (2) is a simple, one-instruction step. To set the MSB of each bit-pair to the correct value, it's only necessary to shift each bit of [AX] one (1) position to the left. This is done by using

```
shl   bx,ax
```

Step three (3) is equally simple. The LSB values are stored in [BX] and the MSB values in [AX]. To get the final result we only need to add them together. That's done by this line of code:

```
add   ax,bx
```

Before returning to the caller, the extra register needed, [BX], is restored.

```
Version 2:

Convert proc near
                                    ;clocks
   push  bx                         ;15
   xor   ah,ah                      ; 3
   mov   bl,al                      ; 2
   and   al,00000001b               ; 4
Second_Bit:
   shr   bl,01                      ; 2
   jnc   Third_Bit                  ;16 if jump, 4 if not
   add   ax,0000000000000100b       ; 4
Third_Bit:
   shr   bl,01                      ; 2
   jz    Fourth_Bit                 ;16 if jump, 4 if not
   add   ax,0000000000010000b       ; 4

      (and so forth)
```

Analysis

As I said, I'm after speed in this routine. So, a little more than a month after doing the original code, I searched for ways to improve it.

After looking at the code for just a few minutes, I realized one way I could gain some speed. Replace all TEST instructions with a shift instruction: SHR specifically. The obvious time savings is two (2) clocks: 5 clocks for TEST vs. 2 clocks for SHR.

For those of you not familiar with the technique of using shift instructions in place of TEST, let me explain. If the value of a register doesn't need to be retained (as in this case), use a shift or rotate instruction to set the carry flag rather than using TEST to set the zero flag. There are times, though, when this does not work (e.g., testing the state of more than 1 bit at a time). Obviously, the technique is most useful when testing either the MSB or LSB of byte or word values.

Since in this particular application it's necessary to look at each bit only one (1) time, there's no need to retain it.

Comparison of the clock cycles shows the gain in speed.

The number of clocks it takes to complete the routine is determined by performing a little addition, both for "best case" (no conditional jumps taken—all bits 0) and "worst case" (jumps taken—all bits 1) conditions.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Get a Grip on Assembly Language.

The Award Winning Visible Computer: 8088.

Assembly language programming is far from easy, but you don't have to be a genius to learn it. Let your PC teach it to you.

The Visible Computer: 8088 is an award-winning mix of book and software for mastering the elusive skills of assembly language.

It's an animated simulation of the PC's microprocessor that takes you inside the 8088 as it executes programs. You'll see how instructions are executed, not just the result.

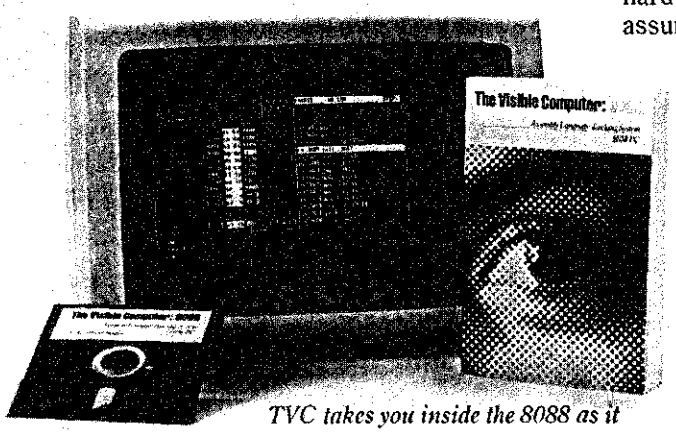
With its intuitive "moving bar" instruction display, memory windows, bulletproof beginner's mode, and more than 30 commands and functions, The Visible Computer is a powerful instrument for inspecting your own programs (or BIOS routines or anything else you can get into memory). You'll be using it as a debugging tool for years to come.

It's a book. The 350 page manual is more than simply instructions on working the simulator – it's a hard-working, assume-nothing tutorial on the fundamental concepts of assembly language. A lot of people think it's the best book on assembly language ever written.

PC Tech Journal
"Program of the Month"

"(The Visible Computer) is at once useful and educational, offering an interactive debugger and a self-paced course in 8088 architecture and assembly language... A considerable value."

The Visible Computer: 8088 for IBM PC/XT/AT and true compatibles.



TVC takes you inside the 8088 as it executes programs.

$49.95 Copy Protected

$79.95 Unprotected

If your dealer doesn't have it, order direct: Software Masters, 2714 Finfeather, Bryan, TX 77801 (409) 822-9490. Please enclose $3.00 shipping. Bank cards accepted.

Software Masters™

Circle Number 32 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

best case: 150 clocks (no jumps)
worst case: 206 clocks (all jumps)
average case: 178 clocks (half jumps)

Now the clock cycles for Version 2:

best case: 131 clocks (no jumps)
worst case: 187 clocks (all jumps)
average case: 159 clocks (half jumps)

The "best case" and "worst case" results are important, but for now a comparison of both "average cases" shows the new routine has a 10.7% speed advantage.

Version 1 average case: 178 clocks
Version 2 average case: 159 clocks
difference: 19 clocks

Further Analysis

Such a gain is significant, to be sure, especially since the routine will be called thousands of times during execution of the main program. But can even more improvements in speed be made?

The use of conditional jump instructions is a boon to programming, but when the jump is made, extra clock cycles are needed. If only there were a way to eliminate their use in this routine . . .

Use of a shift instruction gave me an idea. If SHR allows one to set the carry flag, why not use a rotate instruction to recover the state of the carry flag? Here's what I mean:

```
Assume [AL] = 10100001, [carry flag] = 0,
            [BL] = 00110010
shr  al,01
then  [AL] = 01010000, [carry flag] = 1,
            [BL] unchanged
if we then
rcr  bl,01
then  [AL] unchanged, [carry flag] = 0,
            [BL] = 10011001
```

The routine now accomplishes its task by a one (1) step process rather than three (3) as required by the original. (I choose not to think of repetitious code as multiple steps. But it's only a matter of personal preference.) By first duplicating the contents of [AL] (the passed byte) into both [BL] and [BH], you can get the desired result by using the SHR and RCR instructions.

To prove that the gain in speed was worth the re-coding, here is a partial listing of the new code along with the clock cycles:

```
Version 3:

Convert proc near
                                        ;clocks
  push  bx                              ;15
  mov   bl,al                           ; 2
  mov   bh,al                           ; 2
            ;First_Bit
```


ED/ASM-86

The first truly integrated Editor, Assembler, Debugger, and Linker is now available from Oliver Computing Company.

*** The Editor supports block move, block copy, partial save and partial load insertion. All text is tokenized for both efficiency of storage and very fast assembly time. FAST screen I/O and separate editor/debug windows.**

*** The Assembler can be immediately invoked after any editting command. Output can be directed into memory, to a .COM file, .EXE, or to ED/ASM-86's own efficient .LNK file format. 8087/186/286 instructions supported. Macros and most standard pseudo-ops are supported.**
UNIQUE to ED/ASM is its structured assembly language capability. No more wasted labels and time! Multiple segmented programs can be be sent to .COM files.

*** The built in debugger supports symbolic disassembly, integration with the assembler, as well as disassembly of 8087/186/286 instructions. Display 8087 registers in scientific decimal format. SPLIT SCREEN debugging with a dynamic register window is easily invoked. Disassemble programs to disk file for easier analysis. "Single-step" data editting supported for bytes, words, double words, and 8087 floating constants in memory. You can "patch" assemble a line at a time with symbols from previous assembly (even define symbols as you go!)**

*** The linker supports ED/ASM-86 link files, which can be created from virtually any valid ED/ASM-86 source file, even lines with multiple external symbols in expressions. (Note: not compatible with usual link format)**

*** And many more features too extensive to be listed here.**

All of these features exist in one integrated program. A typical development cycle with ED/ASM-86 consists of entering a program with the editor, or editting an existing program, typing "A:,-IM" to assemble into memory, going into the debugger to immediately test the program. Then you go back to the editor for the next cycle.
ED/ASM was developed for Oliver Computing's own use, and has been used for over a year to enhance itself!

ED/ASM-86 is only $95. Send check or M.O. to:
Oliver Computing Company, P.O. Box 90140, Indianapolis, IN 46290 { (317) 849-4450 }
for immediate delivery via UPS.

Circle 15 on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
shr     bl,1                    ; 2
rcr     ax,1                    ; 2
shr     bh,1                    ; 2
rcr     ax,1                    ; 2
                ;Second_Bit
shr     bl,1                    ; 2
rcr     ax,1                    ; 2
shr     bh,1                    ; 2
rcr     ax,1                    ; 2
                ;Third_Bit
shr     bl,1                    ; 2
rcr     ax,1                    ; 2
shr     bh,1                    ; 2
rcr     ax,1                    ; 2

   (and so forth)
```

For the recoded routine, there is no "best case," "worst case" situation, just an "only-case." Addition gives the number of clocks required by the routine, no matter what the state of each bit.

only case: 111 clocks

The current code shows a 37.6% advantage in speed when compared to the "average case" circumstance of the original routine. And if compared to the original "worst case," the improvement in speed is a whopping 46.1%.

JUMP START Boots IBM Faster for $79.95

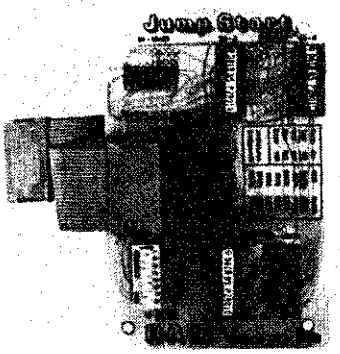
CUTS BOOT TIME
Reduces 640K system boot time to 5 seconds with memory test disabled or 10 seconds enabled (DIP switch options).

PAYS FOR ITSELF QUICKLY
Time savings for $30,000/year employee who reboots 20 times per day recovers entire cost in 3 weeks.

TAKES NO SLOTS
IBM ROM plugs into Jump Start and board replaces ROM in socket. Original BIOS ROM accessible via DIP switch option.

for PC1, PC2, or XT

NEW!
5000 + LPM compiler with macros call for info



JS/PC1
for original 16K PC.
JS/PC2
for 64K PC and 16K PC with hard disk via BIOS change.
JS/XT
for XT and Portable.

DAIR Computer Systems
3440 Kenneth Drive Palo Alto, CA 94303
415 - 494 - 7081

Circle 5 on the reader service card

More Analysis (for size)

At this point, I think it's safe to say that most programmers would be finished, saying the routine is OPTIMIZED. Not me! I'm not easy to satisfy. Even though the idea here is to make speed improvements, it's also a good idea to optimize for size when you can. Assembly language programs don't take up much memory when compared to high-level languages, but . . .

Although I haven't provided a listing showing the number of bytes needed by each version of the routines, the Microsoft Assembler was nice enough to provide the information in the Convert*.LST output. The respective listings have the following sizes:

Version 1: 70 bytes
Version 2: 65 bytes
Version 3: 70 bytes

As you can see, the smallest version is the second, but by only 5 bytes: hardly worth it. To my way of thinking, the gain in speed with Version 3 is well worth the extra 5 bytes.

But don't despair. If a size savings is needed, another change can be made to accomplish it. Much of that code can be eliminated by adding a LOOP instruction and using the [CX] register as a counter. The listing is the result with a size of only 21 bytes, giving a reduction of 49 bytes over the original: a 70% savings.

However, use of a LOOP increases the time needed for the routine to execute.

```
Version 4

Convert proc near
                                ;clocks
  push  bx                      ;15
  push  cx                      ;15
  mov   bl,al                   ; 2
  mov   bh,al                   ; 2
  mov   cx,0008                 ; 4

Move_Bits:
  shr   bl,1                    ; 2
  rcr   ax,1                    ; 2
  shr   bh,1                    ; 2
  rcr   ax,1                    ; 2
  loop Move_Bits                ;17 if loop, 5 if not

  pop   cx                      ;12
  pop   bx                      ;12
  ret                           ;16 (intra-segment)
```

The new "only case" is now slower than the original "average case" by 49.4%. Since this particular routine is called repeatedly by the main program, I'm unwilling to exchange the loss of speed for a gain of memory space. I would look at the program's other routines for space optimization.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Final Analysis

Code optimization comes with experience. The more you program, the more you study others' code, the more likely you are to produce optimized code.

It also helps to periodically take a look at the "old" code you have written. More times than not, you will find such reviews provide new insight into ways of improving the original. I find that reviews refresh my memory considerably. I "re-learn" useful techniques I've not used for a while. And since I'm continuously learning my craft, recently acquired knowledge can usually be applied to code that can be rewritten for improvements in speed and/or size. Optimized.

There is one (1) last technique I would like to leave with you. If you do much assembly language programming, you realize how slow PUSHes and POPs are to execute (15 and 12 clocks, respectively). Whenever possible, move the contents of registers that must be saved into unused registers for later retrieval.

In the case of the Convert routine, [BX] is the only register destroyed by the routine. As it turned out in my program, the [SI] register is changed immediately after program execution returns from the Convert routine. Therefore, to save 25 clocks each time Convert is called, I simply move the contents of [BX] into [SI] upon entering the routine and return the value just before exiting. Making that simple change to Version 3, (my current choice for best-of-the-lot), gives an improvement of 50.6% over the original Version 1 "average case." □

References:

Scanlon, Leo J. and R. J. Brady. *IBM PC Assembly Language*. 1983.

Rector, R. and G. Alexy. *The 8086 Book*. Osborne/McGraw-Hill, 1980.

Michael Hoyt is a contract programming consultant and freelance writer who frequently contributes to Programmer's Journal. *Readers can write to him at*

Rt. 2, Box 65
Solsberry, IN 47459

Excellence

In your job, it depends on having the best tools available at your disposal. With such tools, your productivity increases and your work becomes easier.

Wisely, you keep a sharp eye open for products using the latest technology.

You have now located the source of advanced debugging technology for PC-DOS.
More powerful debugging software is not available anywhere...at any price. Yet the cost is affordable to even the smallest budget.

DSD-86...New and innovative design combines the most sophisticated user interface with the most flexible display to create a new generation of debugging technology for the IBM PC. Only 69.95

SoftAdvances

P.O. Box 49473 - Austin, Texas 78765 - (512) 478-4763

Visa & Mastercard Accepted.
Please include 4.00 for shipping and handling.

Circle Number 33 on the Reader Service Card

Famous Computer Personalities #441 - 449
A small group of early programmers prepares to execute the first conversion routine.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MEMORY RESIDENT CONTROL OVER PRINT SCREEN FUNCTIONS

by Steven Baker

I'll admit it. I program in "C", Pascal, FORTRAN, and BASIC, but I still prefer the control and speed provided by assembly language. The example I'd like to illustrate here, a memory resident utility, shows why: It's small, fast, and deals directly with the 8086/88 and hardware. This program was written in response to requests from several of my clients to correct deficiencies in the IBM PC's own routines.

The IBM Print Screen Function

Depending on one's perspective, the IBM PC comes with either a wonderful or a horrible feature called Print Screen. The ROM BIOS keyboard interrupt routine scans for keystrokes, and if the Shift key and the PrintScreen key are held down at the same time, the keyboard handler calls a Print Screen function. This is just fine if you hit those keys on purpose, but that PrintScreen key is mighty close to the tiny Return key on the IBM. Let's say it was an accident, and let's hope you have a printer connected, 'cause that thoughtful Print Screen routine will wait near forever. If no printer is at hand, you can always reboot and lose the last 30 minutes of that report you've been working on. Oh, I know this has never happened to you, right? With a printer, we only have to turn it on and wait however long it takes to print out the darned page. Let's hope we weren't in the graphics screen mode because the Print Screen function assumes everyone has an IBM graphics printer, right! Otherwise, our printer gets tons of gibberish, too.

PSOFF.COM

The program PSOFF enables or disables the built in Print Screen routine. It was written for assembly into a .COM file. The program was written as a memory resident procedure, which I usually install in a batch file for my clients. For example, an AUTOEXEC.bat file might contain the following:

```
mode com1:,9600,8,1,p mode
lpt:=com1: psoff off
```

This would set up a serial printer as COM1 and disable the Print Screen function on boot up. At any time, PSOFF may be invoked with either the ON or OFF option to disable or reenable the Print Screen function. For example:

```
A>psoff on
```

would enable this function at some later time.

How It Works

The default Print Screen function is set up when the PC ROM BIOS boots up. The ROM BIOS installs the routine as interrupt vector 5. Our own PSOFF will be loaded into memory and will install a replacement routine at this interrupt vector and stay memory resident. When PSOFF is invoked, the ON/OFF parameter will be scanned to determine whether PrintScreen should be turned on or off. Once installed, if we invoke PSOFF again, the program must check to see whether it is already installed in memory. If it's already there, the program just changes the ON/OFF setting and exits without staying resident. If the proper ON/OFF parameter is not found, like any good program PSOFF gives a help message describing its use.

Operating Systems and Assemblers

I write programs to run under both MS DOS by Microsoft and CP/M 86 (and now Concurrent DOS) by Digital Research. I have therefore structured the code with equates to assemble and run under DOS or CP/M using either system's usual assembler. Using MASM (Microsoft's assembler), I have created a series of macros that handle the syntax differences between these two families of assemblers. For development time, ASM86 and GENCMD (DR's absolute assembler and loader) are much faster for smaller assembly language programs than MASM and LINK (Microsoft's relocatable assembler and linker)—perhaps the subject for some future article.

The Interrupt Handler

PSOFF has two basic parts, the initialization code and the interrupt handler itself. In this example, almost all of the code is for initialization. The interrupt handler at NEWINT5 is only six bytes long. It has two modes, either on or off. When PrintScreen is OFF, the interrupt handler consists of just a single IRET instruction (return from interrupt) to complete the function. When PrintScreen is ON, the IRET is replaced by a NOP followed by a JMP FAR instruction to the original interrupt handler. The memory resident part of this routine is only about 40 bytes long.

Initialization Code for Installing Our Interrupt

The initialization code will be used once and then discarded. It starts at the label INIT. The INIT routine goes through a series of small inline routines that consist of the following:

init:	set up segment registers and stack
getparm:	test if proper parameters were given
movint:	if good parameters, copy existing PrintScreen to our own code

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

testint: test that the PSOFF is not already installed

change: if PSOFF already installed, then just change its ON/OFF toggle, tell user current status, and exit

setint: if PSOFF not installed, then install our new INT 5 vector, tell user current status, exit, and stay resident

done: exit and stay resident

The subroutines consist of the following:

bdos: call DOS whether CP/M 86 or MS DOS

pchar: print a single CHAR to the screen

pmess: print a string terminated by binary 0 to the screen

crlf: send a Carriage Return, Line Feed to the screen

saystat: tell user status of Print Screen function

As a matter of course, I always set up the segment residents as I desire and set up my own local stack. Since we are writing a COM file, all the segment registers will be set to CS. Next, we get the parameter (GETPARM) if one was given from the command line. The first two parameters will be capitalized and placed in the file control blocks at 5ch and 6ch by COMMAND.COM. A copy of the raw command line is also available at the default disk transfer address at 80h (no capitalization). If no parameters were given, we first give help before aborting.

The code next tests the first parameter (TESTPARM) for a match with either ON or OFF. The File Control Block is easier to use for testing because the parameters have already been capitalized. For comparison, I have used the string compare operations using SI and DI. If we have a match, we continue; otherwise, we give a help message and exit doing nothing.

The next step is to move the existing INT 5 vector into our own code (MOVINT), again using repeat string operations. The DS register is temporarily set to 0, pointing to the interrupt vector table for this operation, and then restored. Now we compare the existing INT 5 vector to see if PSOFF may already be installed. If PSOFF is already there, we just change the status (CHANGE) by writing either an IRET or a NOP to the interrupt handler's first instruction and exit.

If PSOFF is not already installed, we first set up INT 5 to point to our own code (SETINT). I could have used an MS DOS system call for this function; however, this would not have worked under CP/M 86. For safety's sake, the 8086 interrupts must be temporarily turned off when you are changing the value of any of the interrupt vectors. We finally tell the user the current status and exit staying memory resident. Under MS DOS, the DX register is pointed 1 byte past the code to remain memory resident, and an INT 27h is issued. This differs from the procedure required under CP/M 86. Under MS DOS, only about 40 bytes need to remain memory resident. This includes our 6-byte interrupt handler and a version string message.

The subroutines are fairly simple, with all Character output to the screen going through the function PCHAR. In this way, on a larger project, one could change from screen output through DOS to using either the ROM BIOS or direct screen writes by changing only one function. Since speed is no issue for this program, I have used DOS for character i/o.

A number of enhancements could be added to this simple program. For one, the screen RAM could be saved and the user asked if he or she really wanted to Print the Screen (oh my, I didn't mean to hit that key after all). If the user answers negative, the screen RAM could be restored and control returned with nothing changed. The Print Screen routine could prompt the user to save the screen to a disk file. If you were in the graphics screen mode, the routine could print a graphics dump of the screen to a dot matrix printer or file of your choosing. □

Code Follows

A DREAD-LOCKED KEYBOARD

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

(1)

```
        TITLE     'Print Screen Off Routine for MSDOS 4-3-85'
;         pagewidth       96              ;for CPM86
;
;       Print Screen Off Routine for MSDOS or CP/M86
;       Author:         M. Steven Baker
;       Revision date   March 25, 1985
;       Last revision   April 1, 1985
;
;       Purpose:
;               installs and stays resident in DOS to revector
;               the print screen routine to an innocuous IRET
;
;       Operation:
;               This program must be run first to disable the
;               standard Print Screen Routine for PCDOS
;               which revectors the INT5 to this new code
;               This code stays resident until rebooting
;               to allow a user to either turn ON or OFF
;               the print screen routine
;
;       GENCMD PSOFF 8080  ;if using DR's ASM86
;
;       E Q U A T E S
;
cr      equ     0dh
lf      equ     0ah
;
false   equ     0
true    equ     not false
;
MSDOS   equ     true
CPM86   equ     not MSDOS
;
MASM    equ     true            ;using Microsoft or compatible
                                 assembler
ASM86   equ     NOT MASM        ;using Digital Research assembler
;
        if      MASM            ;use Macros for DR ASM86 syntax

jmps    macro   dummy           ;Jump SHORT
        jmp     short   dummy
        endm
;
rs      macro   count           ;DR's reserved storage
        db      count dup(?)
        endm

cseg    macro                   ;DR's begin Code segment
        CODE    segment
        assume  cs:CODE,ds:CODE
        endm

        ENDIF   ;MASM

        cseg

fcb     equ     05ch            ;file control block for parameters
;

        ORG     0100h
;
; Sign on message
;
begin:  jmp     init            ;jump to initialization code
vernm   db      'Print Screen On/Off Version 1.0  4-3-85',0
;
```

(2)

```
newint5 db      90h             ;NOP space for our IRET
        db      0eah
int5    dw      0,0             ; kluge jmpf
;
;
;       I N I T
;       initialization code for installing our interrupt
;
init:   mov     ax,cs           ;setup segment registers
        mov     ds,ax
        mov     es,ax
        mov     ss,ax
        MOV     SP,Offset stack
;
        CALL    crlf
;
; test if proper parameters are there
;
getparm:
        mov     si,fcb+1
        mov     al,[si]
        cmp     al,' '
        jne     testparm        ;if no parameters, then give help
help:   jmp     givehelp
;
testparm:
        mov     di,offset on_stg ;point to ON string
        mov     cx,4            ;compare four bytes
        repe    cmpsb           ;comparison
        jne     tparm2
        mov     byte ptr newint5,90h    ;put NOP at newint5
        jmps    movint
;
tparm2: mov     si,fcb+1        ;point to parameter
        mov     di,offset off_stg ;point to OFF string
        mov     cx,4            ;compare 4 bytes
        repe    cmpsb           ;compare them
        jne     help
        mov     byte ptr newint5,0cfh   ;put IRET opcode at newint5
;
; copy print screen interrupt vector to our JUMP FAR return
; used to return from our checking code
;
movint: xor     ax,ax           ;zero AX
        mov     ds,ax           ;set DS to segment 0
        cld                     ;set forward direction
        mov     si,14h          ;pointer to int5
        mov     di,offset int5  ;pointer to our JUMP FAR code
        mov     ax,cs
        mov     es,ax           ;set ES to our code segment
        movsw                   ;move offset
        movsw                   ;move segment
;
; now test that we have not already previously installed this
; at the print screen interrupt
;
testint:
        mov     si,14h          ;pointer to int5
        lodsw                   ;get offset to AX
        mov     dx,ax           ;save offset to DX
        lodsw                   ;get segment to AX
        mov     es,ax           ;temporrarily store it in ES
        mov     ax,cs           ;restore our DS register
        mov     ds,ax
        mov     ax,offset newint5
        cmp     ax,dx           ;offsets are not equal
        jne     setint
;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        mov     si,offset vernm ;point to version name
        mov     di,si           ;in both SI and DI
        mov     cx,(offset newint5)-(offset vernm)
        repe    cmpsb           ;compare them for equal
        jne     setint
;
change: mov     si,offset newint5
        mov     di,si
        movsb                   ; and CHANGE it
        call    saystat         ;give user current status
        jmp     exit
;
; now replace INT5 interrupt vector with pointer to our code
;
setint: mov     ax,cs
        mov     ds,ax   ;setup DS = CS
        mov     bx,ax   ;save our code segment in BX
        xor     ax,ax   ;zero AX
        mov     es,ax   ;setup ES as segment 0
        mov     di,14h  ;point to INT5 vector
;
        cld             ;set forward direction
        cli             ;shut hardware interrupts off
;
        mov     ax,offset newint5       ;setup our int5
        stosw
        mov     ax,bx   ;now set our code segment
        stosw           ;and store it
;
inton:  sti             ;turn back on interrupts
                        ;vector has now been replaced
;
        mov     ax,cs
        mov     ds,ax
;
        call    saystat         ;tell them status
;
;
done:
        if      cpm86
; exit and stay resident under CPM86
        mov     cl,0    ;setup for exit
        mov     dl,1    ;stay resident
        call    bdos
        endif

        if      msdos
; terminate but stay resident under dos
        mov     dx,(offset init)+1
        int     27h
        endif
;
; exit but don't stay resident
;
givehelp:
        mov     si,offset helpmsg       ;give help message
        call    pmess
                                        ; and fall thru to exit
exit:
        mov     ax,0
        mov     dx,ax           ;set DX =0 for CPM86 exit
        call    bdos
;
;       S A Y S T A T U S
;       say status of Print Screen Function
;       ENTRY   none
;       EXIT    AX and SI are destroyed
```

```
;
saystat:
        mov     si,offset endmsg        ;tell them we're done
        call    pmess
        mov     al,byte ptr newint5     ;get value either NOP or IRET
        mov     si,offset onmess
        cmp     al,90h
        je      saystat_2
;
        mov     si,offset offmess
        cmp     al,0cfh
        je      saystat_2
        mov     si,offset badmsg
saystat_2:
        call    pmess
        call    crlf
        ret
;
; DOS interface
;
crlf:   MOV     AL,cr
        CALL    pchar
        MOV     AL,1f
;
pchar:  PUSH    DX
        PUSH    BX
        MOV     DL,AL
        MOV     ah,2
        call    bdos
        POP     BX
        POP     DX
        RET
;
;       P M E S S
;       print message to screen
;       Entry   SI = pointer to message terminated by null byte
;       Exit    AL and SI destroyed
pmess:  lodsb                   ;get byte
        or      al,al           ; is it zero??
        jnz     pmess2
        ret
pmess2: push    si              ;save pointer
        call    pchar           ;send character
        pop     si              ;restore pointer
        jmps    pmess           ; and continue

bdos:
        if      cpm86
        push    es              ;preserve ES
        push    cx              ;save CX
        mov     cl,ah           ;for cpm86
        int     0e0h
        pop     cx              ;restore registers
        pop     es
        endif
;
        if      msdos
        int     21h
        endif
        ret
;
        if      ASM86           ;if using DR's ASM86
        datast  equ     $
        DSEG
        org     offset datast
        ENDIF   ;asm86
;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C Scientific Library

$245

ADVANCED MATRIX COMPUTATIONS
SOLVE SYSTEMS OF DIFFERENTIAL EQUATIONS
QUADRATURE AND DIFFERENTIATION, INTERPOLATION
STATISTICS, PROBABILITY, AND DATA ANALYSIS
SOLVING NON-LINEAR EQUATIONS
COMPLEX VARIABLES AND POLYNOMINALS
FINANCE AND CALENDAR FUNCTIONS
UTILITIES AND GRAPHICS (for Z100 and IBM/PC)

Manual and source code examples of function usage are included. Libraries contain large and small model, linkable functions, requiring the Computer Innovations C86 Compiler v2.2+, MSDOS (PCDOS) v2.+, and 8087 coprocessor. Most disk formats available. 25% discount on CSL when ordered with the C86 compiler and a personal check. Manual and demonstration disk $75, refundable on purchase of CSL.

COMPILERS: Computer Innvoations C Compiler	$298
C Training System with Interpreter	$125
The Software Toolworks C & MATHPAK Comb	$ 75

Eigenware Technologies
13090 La Vista Drive
Saratoga, CA 95070
(408) 867-1184

Circle Number 29 on the Reader Service Card

⑤

```
; message texts
;
helpmsg db      'USAGE as follows:',cr,lf
        db      ' A>PSCREEN ON (to ENABLE print screen function)'
        db      cr,lf
        db      ' A>PSCREEN OFF (to DISABLE print screen
        db      cr,lf,0                                 function)'
;
endmsg  db      'Print Screen Function is ',0
onmess  db      'ENABLED - '
on_stg  db      'ON  ',0
offmess db      'DISABLED - '
off_stg db      'OFF ',0
;
badmsg  db      '*** CORRUPTED OR DAMAGED ***'
        DB      cr,lf,'PLEASE REBOOT SYSTEM'
        db      cr,lf,0
;
;
        RS      100
stack   dw      0
;
intend  equ     $

        if      MASM
        CODE    ends
        ENDIF

        end
```


C CODE FOR THE PC

source code, of course

Concurrent C	$45
Coder's Prolog in C	$45
LEX	$25
YACC & PREP	$25
Small-C compiler for 8088	$20
tiny-c interpreter & shell	$20
Xlisp 1.5a & tiny-Prolog	$20
C Tools	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409
(512) 258-0785

Free shipping on prepaid orders — No credit cards

Circle Number 30 on the Reader Service Card

INTRODUCING DATALIGHT C

The Datalight C Compiler for MSDOS is a full C with all K&R constructs, including bitfields, plus the version 7 extensions. Other features of the compiler are:

$60

- Produces object files (.obj) so just the MSDOS linker is required.
- Floating point performed with 8087 or automatic software floating.
- Over 100 compact library functions with source.
- Compatible with the Lattice C compiler.
- Runs on IBM-PC, and compatibles, running MSDOS 2.0 or later.
- Complete, easy-to-read users' manual with index.
- Highly optimized code for production quality programs.

Datalight
11557 8th Ave. N.E.
Seattle, Washington 98125
(206) 367-1803

Outside USA add $10 shipping. Washington State residents add 7.9% sales tax. VISA and MasterCard accepted.

IBM-PC, a trademark of IBM; MSDOS, a trademark of Microsoft Corp.; Lattice C, a trademark of Lattice Corp.

Circle Number 7 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

STRUCTURING BASIC OUTPUT WITH FIELD STATEMENTS

by Bernard H. Robinson, Jr.

It may come as a surprise to some to learn that BASIC FIELD statements can be used to structure ASCII output to the screen or printer, as well as for their more usual application to handling data for random access files. Although beginners should be able to follow this discussion, it will be easier for readers with some familiarity with random access file programming. Some of the techniques presented here are quite useful when working with files, but we will concentrate on screen and printer output. There are many different approaches to structuring output data in BASIC, and, as is so often the case, the programmer needs to consider the advantages and disadvantages of the various approaches in order to choose among them.

This discussion is specifically for the IBM PC BASIC interpreter as supplied with IBM DOS 2.0. Some modifications may be necessary for other BASIC interpreters. BASIC compilers will certainly require some modifications, but I am not sufficiently familiar with BASIC compilers to discuss them.

The use of FIELD statements should be considered whenever the programmer wants to keep tight control over the position of data on the screen or on the printed page, as for columns of data under headings. Using FIELD statements makes it easy to insert or change data in a particular column of an output line without disturbing data in other columns of the line. Each buffer that is set up in a program for use in formatting requires 188 bytes of memory plus the length of the buffer in bytes. Because one buffer can be used repeatedly within a program for a variety of different formats, this memory requirement is usually insignificant. A small amount of additional memory will be used to hold the FIELD statements themselves.

The basic technique is simple. A file buffer must be set up just as it would be to output data to a file. FIELD statements are then used to structure the file buffer. Both of these steps can be done quite easily as explained below. The structures set up by FIELD statements act as templates that define locations in which data will be printed on the output line. Any number of different structures can be defined for a single buffer.

Example Program

Figure 1 is a simple example program, TFIELD.BAS, and Figure 2 shows the output from this program. References to line numbers below refer to lines of Figure 1. The program illustrates the formation of numerical output into columns. Four structures for the file buffer were useful. One structure positions the words for the first line of the column headings, the second positions the words for the second line of the column headings, the third positions dashes used to separate the heading from the numerical data, and the fourth positions the numerical data under the headings. All of these structure definitions are available at the same time, but only the appropriate one is selected for use for a given line of output. The output in this case is to the screen, but it could have been to a file, or a printer.

Establishing a File Buffer

A file buffer is established by executing a normal OPEN statement as for opening a random access file on the disk (line 5010). As usual, the OPEN statement must contain a file reference number and the FIELD statements that structure the buffer must use the same reference number. The OPEN statement actually opens a file on the disk, but since little data is written to the file and it is erased by the program when no longer needed, this is no problem. The name of the dummy file to be opened, "QQQ.WWW", is assigned to the variable "XFNAME$" in line 140. Note the "X" in "XFNAME$"; BASIC reserves all variable names starting with "FN" for user defined functions, and if we had used simply "FNAME$", BASIC would have favored us with a cryptic error message that might have caused us much frustration to troubleshoot. The file name, "QQQ.WWW", was chosen as not likely to coincide with any already on the disk.

The "LEN = 48" portion of the OPEN statement defines the length of the buffer established by the execution of the OPEN statement. It must be at least as long as the total width implied by the FIELD statements. A maximum width of 79 is convenient for writing to the screen. Unless special steps are taken, a width of 80 results in automatic line feeds at the end of each line, causing a blank line between each line of the output. For output to a printer or file, wider buffers may be desired. If the buffer width for any OPEN statement in a program is larger than 128, BASIC must be configured for this by the inclusion of "/S:nnn" in the DOS command to start BASIC. For example, if the maximum width is 132 bytes, BASIC must be called up with the DOS command:

BASIC /S:132

Multiple Field Statements

Much of the usefulness of this method of formatting derives from the use of multiple FIELD statements. The IBM BASIC manual states that any number of FIELD statements may be executed on a given file number, that each one redefines the buffer structure from its beginning, and that all that have been executed remain in effect at the same time. The best way to grasp the meaning of these statements is to consider a

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

few examples. The first FIELD statement, line 5020, sets up the format for the first line of the heading of the output (see Figure 2 for the printed output). The individual fields in the FIELD statement are identified by the names (string variable names) assigned in the statement. The first five spaces is designated as a field named "ITEM$"; the next 10 spaces is defined as a field named "START$", etc. The "ITEM$" field is designed to contain the word "ITEM", the "START$" field to hold the word "STARTING", etc. I chose to lay out these fields so that the "RSET" statement, which right-justifies data in the fields, would be used in inserting the data into the fields. It is usually helpful to lay out the fields with pencil on a sheet of paper with numbered columns.

Line 5030 redefines the buffer for the second line of the column headings. According to the rules given above, each FIELD statement must redefine the buffer from the first position in the buffer; however, the definition need not extend to the end of the buffer. Note that in line 5030 different names from those used in line 5020 are given to the fields. Both FIELD statements are in effect at the same time, so any field name in any FIELD statement can be used at will. If we RSET data into START$ and then RSET data into DATE1$, the data for DATE1$ would overwrite part of the data put into START$ because the definitions of the two fields overlap in their respective FIELD statements. The normal way to use the buffer is to load it for one structure at a time and print the data before loading the next.

Lines 5060 through 5080 illustrate how multiple lines are used to define one template. Line 5060 defines the first 15 spaces of the line of numerical data. In order for line 5070 to take the definition further, it must start with a dummy field (we call it "Z$") of 15 spaces to account for the 15 spaces defined in line 5040. The rest of line 5070 defines 10 more spaces of the template. Line 5080 starts with a dummy field of 25 spaces to account for the spaces defined in lines 5060 and 5070. Note that we can use the same name, "Z$", for the dummy field repeatedly. We can do this because BASIC remembers only the last field assignment when a name is used more than once, but it retains the field spacing set up by the earlier usage. We do not plan to use Z$ in statements to load data into or out of the buffer, so we do not care that BASIC forgets the earlier field assignment to it.

Line 5090 assigns all 48 spaces of the buffer to the field name "TF$". After the buffer has been loaded, printing "TF$" will print the entire contents of the buffer.

The Effect of Opening and Closing Files

It was pointed out that it is necessary to open a disk file to establish a buffer. A natural question is whether the file must remain open in order to use the buffer. The answer is no. The file must be open while the FIELD statements are being executed, but after that, closing the file does not alter either the structure or the contents of the buffer. Note that in line 5090 the

Instant-C™:
The *Fastest* Interpreter for C

Runs your programs 50 to 500 times faster than any other C language interpreter.

Any C interpreter can save you compile and link time when developing your programs. But only ***Instant-C*** saves your time by running your program at compiled-code speed.

Fastest Development. A program that runs in one second when compiled with an optimizing compiler runs in two or three seconds with ***Instant-C***. Other interpreters will run the same program in two minutes. Or even ten minutes. Don't trade slow compiling and linking for slow testing and debugging. *Only* ***Instant-C*** *will let you edit, test, and debug at the fastest possible speeds.*

Fastest Testing. ***Instant-C*** immediately executes any C expression, statement, or function call, and display the results. Learn C, or test your programs faster than ever before.

Fastest Debugging. ***Instant-C*** gives you the best source-level debugger for C. Single-step by source statement, or set any number of conditional breakpoints throughout your program. Errors always show the source statements involved. Errors always show the source statements involved. Once you find the problem, test the correction in seconds.

Fastest Programming. ***Instant-C*** can directly generate executable files, supports full K & R standard C, comes with complete library source, and works under PC-DOS, MS-DOS, or CP/M-86. ***Instant-C*** *gives you working, well-tested programs faster than any other programming tool.* Satisfaction guaranteed, or your money back in first 31 days. ***Instant-C*** is $495.

Rational
Systems, Inc.

P.O. Box 480
Natick, MA 01760
(617) 653-6194

Circle Number **20** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

file is closed and the disk file that it created is deleted ("killed"). On the other hand, the opening of a file with the same reference number (or the reopening after it has been closed) does alter the contents of the buffer by filling it with blanks in the form of CHR$(0) characters (not spaces).

Because the opening and closing of a disk file each requires a disk access, it will speed matters up to open the file on a RAM disk.

Multiple Buffers

One buffer will suffice for many applications of this method, but sometimes you may need two or more buffers in use at the same time. BASIC limits the number of data files that may be open at the same time in a program to a default number of three, and a file buffer in use for formatting counts as one of these. BASIC will permit only the numbers 1,2 and 3 to be associated with files and buffers unless the allowable number of files is extended. The number of files can be extended to four if the "/F:4" option is used in the DOS command to call up BASIC. The number can be extended to a maximum of 15 through a CONFIG.SYS file on your boot disk plus the use of the "/F:" switch on the BASIC command.

If your boot disk already has a CONFIG.SYS file on it, check the contents of the file with the DOS command TYPE CONFIG.SYS. If the CONFIG.SYS file has a FILES = nn line in it, then the maximum number of files you can open in BASIC is 4 less than the number specified, up to a maximum of 15 files. Thus, if the CONFIG.SYS file has a line reading FILES = 20, as would very likely be the case if you are using dBASE III, then you could open the maximum number allowed by BASIC (20–4 = 16, which is larger than the 15 maximum allowed by BASIC).

If your boot disk already has a CONFIG.SYS file and you decide to change it, be sure to include everything that is now in the file, changing only the FILES = nn statement, if there is one, or adding the statement if there is none. Otherwise, some of your programs may cease to work. If you are not sure what should be in your CONFIG.SYS file, play safe— make a backup copy of your old CONFIG.SYS file by copying it to some other file name such as CONFIG.BAK before changing it. CONFIG.SYS is simply an ASCII text file and can be created in the same way as a .BAT file. You can use any straight ASCII text editor, EDLIN or the COPY CON: method suggested in the DOS manual, for making AUTO EXEC.BAT files.

Even after changing the CONFIG.SYS file, you must still include the "/F:nn" option in the BASIC command if you want more than 3 files and/or buffers open at once. The number nn of the /F: switch is the actual number of files or buffers that may be in use at once, and the number in the CONFIG.SYS file must be at least 4 greater than that. Remember that the addition of or changes to a CONFIG.SYS file are effective only after the system has been rebooted with the new CONFIG.SYS file on the boot disk.

As pointed out above, a file buffer can have any number of active FIELD statements, so if you are using a buffer in connection with a random access file, it may be possible to share it as an output formatting buffer. The length of the file buffer is critical for the file buffering application, but for output formatting, a length longer than the minimum required does not interfere with its use. You must be sure, however, that the buffer is not needed for both applications simultaneously.

Printing Column Headings

Subroutine 5200 puts data in the buffer for printing the first line of the titles of the output columns. The RSET statement, which right-justifies the data in the buffer field, is used for this. If the buffer structure had been set up for it, the LSET statement could have been used just as well for the titles.

The subroutine at line 5400 sets up and prints the second line of the column headings and the line of dashes that separates them from the numerical data.

Top Quality Programming Tools from the Developers of the TurboPower Utilities

NEW!

Turbo EXTENDER™

Tired of fighting 64K Code and Data Segments?
Bored while waiting for your 10,000 liner to Compile?
Want to optimize those sluggish Overlays?

LARGE CODE MODEL

Write Turbo Pascal programs using all 640K of MSDOS memory, based on any number of separately compiled modules. Provides complete parameter passing using normal Pascal syntax. Heap and Data Segment are shared between all modules. No memory-resident kludges or unnatural parameter passing schemes. Comes with a utility which automatically converts your existing applications.

LARGE DATA ARRAYS

Transparently access 1 and 2 dimensional arrays of any conceivable size and type. Four models support Normal RAM to 640K, Expanded memory (EMS) to 2Meg, Virtual (Disk-based) to 30Meg, and sparse arrays like the most advanced spreadsheets. Comes with a fast full-screen array browser.

MAKE FACILITY

A Unix-like MAKE program that is optimized for the Turbo EXTENDER large code model. Rebuild multi-module programs with no wasted effort.

OVERLAY ANALYST

Perform Static and Dynamic analysis of overlayed Turbo programs. Determine sizes of all procedures in each overlay group. Monitor the running program to find the number of overlay reads, procedure calls, and the load address of all procedures.

AND EVEN MORE!

DISK CACHE can be incorporated in your program to speed up disk reads for data bases, overlays, et al. Multi-file full screen BROWSE works on any text file. Pascal ENCRYPTOR makes your source safe from prying eyes, improves compile speed 15-30% and leaves the code 100% functional. SHELL generator creates fast compiling shells of unexercised code.

Two DSDD disks with complete Source Code, 100 page printed manual, 30 day guarantee!
Requires Turbo Pascal 3.0 and DOS 2.X or 3.X. Runs on IBM PC/XT/AT and compatibles. Call for generic MSDOS support.

$85 complete

Also get the TurboPower Utilities with the acclaimed Pascal Structure Analyzer
Includes a Pretty Printer, Execution Profiler, and powerful Text and Command Automation Tools. With full source $95, executable only $55.

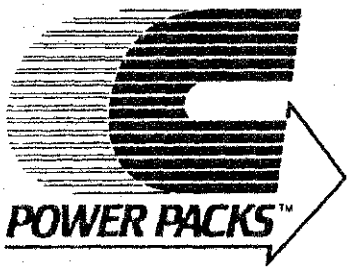
Credit Card Orders only call Toll-free 7 days per week
(US)800-538-8157x830 (CA)800-672-3470x830
PO, COD, Dealers, Questions, Brochures, call or write:

TURBO Power
478 W. Hamilton #196
Campbell, CA 95008
ph. 408-378-3672
M-F 9AM-5PM PST

Circle Number 25 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

YOU NEED A GOOD LIBRARY



COMPLETE SOURCES
NO ROYALTIES

COMPREHENSIVE C Power Packs include over 1000 functions which provide an integrated environment for developing your applications efficiently. "This is a beautifully documented, incredibly comprehensive set of C Function Libraries."
– Dr. Dobb's Journal, July 1984

USEFUL "...can be used as an excellent learning tool for beginning C Programmers..."
– PC User's Group of Colorado, Jan. 1985

FLEXIBLE Most Compilers and all Memory Models supported.

RECOMMENDED "I have no hesitation in recommending it to any programmer interested in producing more applications code, using more of the PC capabilities, in much less time." – Microsystems, Oct. 1984

- **PACK 1: Building Blocks I** $149
  DOS, Keyboard, File, Printer, Video, Async
- **PACK 2: Database** $399
  B-Tree, Virtual Memory, Lists, Variable Records
- **PACK 3: Communications** $149
  Smartmodem™, Xon/Xoff, X-Modem, Modem-7
- **PACK 4: Building Blocks II** $149
  Dates, Textwindows, Menus, Data Compression, Graphics
- **PACK 5: Mathematics I** $99
  Log, Trig, Random, Std Deviation
- **PACK 6: Utilities I (EXE files)** $99
  Arc, Diff, Replace, Scan, Wipe

Master Card/Visa, $7 Shipping, Mass. Sales Tax 5%
ASK FOR FREE DEMO DISKETTE

NOVUM ORGANUM INC.
SOFTWARE HORIZONS INC.
44 Mall Rd. Burlington, MA 01802
(617) 273-4711

Circle Number 23 on the Reader Service Card

A word of caution at this point: Once a FIELD statement has been executed, the variable names assigned to fields may be used on either side of LSET and RSET statements and on the RIGHT side of ASSIGNMENT statements, but must NEVER be used on the LEFT side of ASSIGNMENT statements. If this rule is violated, BASIC changes the variable name from a field designator to a regular string variable and no longer associates it with the field. There is no error message when this happens, and the error can be tough to find. This is also true when FIELD statements are being used for their more customary application to random access files.

Numerical Data Considerations

All numerical values must be in the form of ASCII strings. Usually the numerical values are initially in the form of numerical variables that must be converted to strings. Use the STR$() function to make this conversion. The MKI$(), MKS$(), and MKD$() functions do not produce ASCII strings and must not be used for data that is to go to the screen or a printer.

Be sure that fields for numerical data are long enough to contain the data. If a number is too long to fit into a field, it will be truncated without an error message from BASIC. You may have a clue, however, because the truncation will be on the right end of the number and you will not see two places to the right of the decimal point of dollar and cents amounts.

For neat numerical output, attention must be given to small details. For example, dates in the form "01/05/85" look better if the initial "0" is printed. Also, when a positive number is converted to a string with the STR$() function, a blank space is appended to the front of the string (space for the negative sign, when required). This blank space can spoil the appearance of the output. The function defined in line 120 (for positive numbers only!) and the subroutine at line 7000 are used to make a neat date format.

When numbers have a decimal fraction part, such as dollars and cents, it is highly desirable to print them with the decimal points in alignment. Often the amounts are computed and have more than two places to the right of the decimal point, requiring that the numbers be rounded to two places. A complete discussion of rounding for IBM PC BASIC is a subject that can fill several pages. The subroutine at line 6500 is one that I have developed and found to be very useful for dollar and cents rounding. It requires that the input number be assigned to the double precision variable QR#. Executing subroutine 6500 gives the string QR$ with the number rounded to two places. The string will always contain two figures for the cents, even if the cents part is "0 0", so that when RSET into a field, the decimal points will line up. When rounding, the programmer sometimes wishes to retain the unrounded number for further calculations; on other occasions, he or she may wish to use the rounded number instead for further calculations. Thus the routine at line 6500 provides a string, QR$, with rounded data ready for printing and a double precision variable, QR1#, with the rounded value in it. The original input value is unchanged, so the programmer has a choice of rounded or unrounded numerical variables.

Before using the routine at line 6500 in your programs, I suggest that you test it and see whether it meets your requirements. I have tested it on values from zero to values approximating the national debt, and I am satisfied with the results. As an example of the frustrating situations that are encountered when rounding numbers in BASIC, consider the following: Suppose a program computes a single precision value of .095 (nine and a half cents). If we write QR# = .095 and execute subroutine 6500, we would expect to get .10; however, due to BASIC's binary representation of .095 in single precision, the 6500 subroutine gives the result as .09. Since .095 starts as a single precision number, the double preci-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

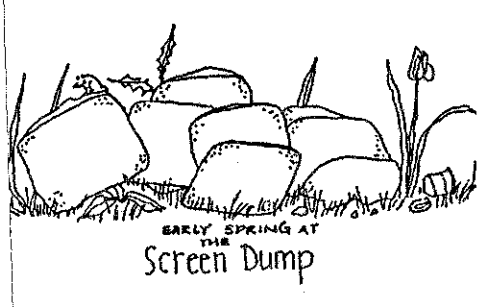
sion value assigned to QR# is .0949 9999880790711#. This is close to .095, but less so that rounding to .09 occurs. On the other hand, if a double precision value of .095# is assigned to QR#, the 6500 subroutine will round it to .10 as expected. The best that I can recommend is to enter all data as double precision data and do all calculations in double precision if you wish to come as close as possible to the results that would be obtained using purely decimal arithmetic.

Moving Numerical Data into the Buffer

Once converted into ASCII strings, numerical data is moved into the buffer in the same manner as were the title strings, except that numerical strings are almost always moved in with the RSET statement to align the decimal points. Subroutine 6000 formats a line of numerical data in the buffer and prints it.

Conclusion

The method described above gives tight control over the format of ASCII data output from a BASIC program and deserves consideration along with the many other methods of formatting BASIC output data. □




```
10 '          Program name: TFIELD.BAS
20 '               Purpose: Demo of formatting with field statements
30 '            Written by:  B. H. Robinson
40 '               Version: 010, October 8,1985
100 DIM D$(3),P$(2),TITLE$(2),U$(4)
110 DEFINT I
120 DEF FNSTRP$(V)=RIGHT$(STR$(V),LEN(STR$(V))-1)
130 XLAR$=STRING$(48,32) ' String of 48 spaces
140 XFNAME$="QQQ.WWW" ' Name of dummy file to open
150 GOSUB 5000 ' Set up field definitions
160 GOSUB 5200 ' Set up and print first line of title
170 GOSUB 5400 ' Set up and print second line of title
180 ' Next, set up and print lines of numerical data
190 ' Input data for line 1
200 ITEM=1 : MONTH1=1 : DAY1=3 : YEAR1 = 1985
210 MONTH2=1 : DAY2=31 : YEAR2=1985
220 LTOT#=0 : AMT#=572.236#
230 GOSUB 6000 ' Format and print first line of data
240 ' Data for 2nd line
250 ITEM=2 : MONTH1=2 : DAY1=4 : YEAR1=1985
260 MONTH2=2 : DAY2=27 : YEAR2=1985 : AMT#=753.762#
270 GOSUB 6000 ' Format and print 2nd line
280 END
5000 ' Set up fields
5010 OPEN XFNAME$ AS #1 LEN = 48
5020 FIELD #1,5 AS ITEM$,10 AS START$, 7 AS EN$, 12 AS PERIOD$, 14 AS ACCUM$
5030 FIELD #1,5 AS NO$, 8 AS DATE1$, 10 AS DATE2$, 11 AS AMOUNT$, 10 AS TOTAL$
5040 FIELD #1, 1 AS Z$,4 AS U$(0),2 AS Z$,8 AS U$(1),2 AS Z$,8 AS U$(2)
5050 FIELD #1, 27 AS Z$,7 AS U$(3),2 AS Z$,12 AS U$(4)
5060 FIELD #1, 4 AS ITNO$,5 AS MO1$,1 AS D$(0),2 AS DA1$,1 AS D$(1),2 AS YR1$
5070 FIELD #1, 15 AS Z$, 4 AS MO2$, 1 AS D$(2),2 AS DA2$,1 AS D$(3),2 AS YR2$
5080 FIELD #1, 25 AS Z$,9 AS PAMT$,14 AS ATOT$
5090 FIELD #1, 48 AS TF$
5100 CLOSE #1 : KILL XFNAME$ : RETURN
5200 ' Set up and print first line of title
5210 RSET ITEM$="ITEM" : RSET START$="STARTING" : RSET EN$="END"
5220 RSET PERIOD$="PERIOD" : RSET ACCUM$="ACCUMULATIVE"
5230 TITLE$(0)=TF$ : PRINT TITLE$(0) : RETURN
5400 ' Setup and print second line of  title
5410 LSET TF$=XLAR$ ' Clear buffer
5420 RSET NO$="NO." : RSET DATE1$="DATE" : RSET DATE2$="DATE"
5430 RSET AMOUNT$="AMOUNT" : RSET TOTAL$="TOTAL"
5440 TITLE$(1)=TF$ : PRINT TITLE$(1)
5450 LSET TF$=XLAR$ : RSET U$(0)="----" : RSET U$(1)="--------"
5460 RSET U$(2)=U$(1) : RSET U$(3)="--------" : RSET U$(4)=STRING$(12,"-")
5470 TITLE$(2)=TF$ : PRINT TF$ : RETURN ' Print separator
6000 ' Set up and print a line of numerical data
6010 LSET TF$=XLAR$ ' Set buffer to all spaces
6020 A$=FNSTRP$(ITEM) : RSET ITNO$=A$
6030 P(0)=MONTH1 : P(1)=DAY1 : P(2)=YEAR1-1900
6040 GOSUB 7000 ' Format date 1
6050 RSET MO1$=P$(0) : RSET DA1$=P$(1) : RSET YR1$=P$(2)
6060 P(0)=MONTH2 : P(1)=DAY2 : P(2)=YEAR2-1900
6070 GOSUB 7000 ' Format date 2
6080 RSET MO2$=P$(0) : RSET DA2$=P$(1) : RSET YR2$=P$(2)
6090 FOR I=0 TO 3 : RSET D$(I)="/" : NEXT I
6100 QR#=AMT# : GOSUB 6500 ' Round and format period amount
6110 RSET PAMT$=QR$ : ACTOT#=LTOT#+QR1#
6120 QR#=ACTOT# : GOSUB 6500 ' Round and format accum. total
6130 RSET ATOT$=QR$ : LTOT#=QR1#
6140 PRINT TF$ : RETURN ' Print the line & return
6500 ' Rounding subroutine
6510 ' Enter with QR#=number to round
6520 QR$=STR$(INT((QR#*100#)+.5#)) : IF LEN(QR$)>2 THEN 6550
6530 IF LEN(QR$)=1 THEN QR$="00" : GOTO 6550
6540 QR$=LEFT$(QR$,1) +"0"+RIGHT$(QR$,1)
6550 QR$=LEFT$(QR$,LEN(QR$)-2)+"."+RIGHT$(QR$,2) : QR1#=VAL(QR$)
6560 RETURN
7000 ' Format date substrings
7010 FOR I=0 TO 2 : P$(I)=FNSTRP$(P(I))
7020 IF LEN(P$(I))=1 THEN P$(I)="0"+P$(I)
7030 NEXT I : RETURN
```

Figure 1

ITEM NO.	STARTING DATE	END DATE	PERIOD AMOUNT	ACCUMULATIVE TOTAL
1	01/03/85	01/31/85	572.24	572.24
2	02/04/85	02/27/85	753.76	1326.00

OUTPUT FROM TFIELD.BAS

Figure 2

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NETWORKING—WHAT'S THE PROBLEM?

by Robert R. Shaffer

Many companies are concentrating on the hardware to enable networking. But is hardware the biggest problem in networking? According to 3Com President L. William Krause, "The hardware is not where the issue is. It's in the software . . . ." In an informal discussion with a leading distributor of Corvus Omninet products in Seattle recently, I learned that "there is a severe lack of true multi-user database and accounting software capable of running under MS DOS on a network." Most of the networking solutions are hardware oriented, tending toward a network file server—one that will interface with DOS and provide RECORD LOCKING. It is accepted that disk locking, volume locking, and file locking are too clumsy to be practical in a multi-user environment.

A growing number of programmers are seeing that the real solution should be within the particular database, accounting, or other application program itself—the program itself must be capable of handling the record locking necessary to prevent multi-user mishaps. Providing the solution within the application itself means that the application can be used on ANY networking hardware that is DOS based. This means a wider market for the product.

Now, Microsoft has come quietly to the aid of programmers who love BASIC and want to get into this multi-user application business! They recently introduced the Microsoft QuickBASIC Compiler. I was quick to grab this new compiler.

Short Lived Disappointment

My first look at Microsoft's new QuickBASIC Compiler was disappointing: I tried to compile a public domain Lunar Lander program. It failed because the new compiler does not support BASIC's BLOAD command. But, when I looked deeper within the manual, I started to get excited about the possibilities of this new compiler—here was an easy way to program multi-user software!

The Solution to the Networking Problem

Microsoft's manual seems to hide the real power in this new complier. In their introduction/overview, Microsoft highlighted the new capability to have 64K of data space AND 64K of code space per module—great for larger applications. And, they pointed out the new ability to use Dynamic Arrays for greater flexibility and more efficient use of memory.

But, wait, it isn't until page 26 of the manual that one learns about their "New BASIC Statements and Functions." There, quietly documented, is the new LOCK . . . UNLOCK statement, which "controls access by other processes to all or part of an opened file."

Also, quietly hidden there, are other goodies: the ability to REDIM dynamic arrays; the SHARED statement, which allows subprograms access to main program variables without passing parameters; and the STATIC statement, which designates variables and arrays as local to a subprogram or function definition. Further, the manual barely acknowledges the power of subprograms vs. main programs. Doesn't Microsoft realize that all these enhancements would be eagerly welcomed by those of us who appreciate modular programming—if only they'd have "tooted their own horn" a little louder!

Generic Calendar Reminder

Developing a program module for a calendar/reminder? Why reinvent the wheel? Software Studios has developed a generic calendar/reminder written in BASICA with the following capabilities:

Full-Month Graphic Display	Full Cursor Control
Reminder Dates Highlighted	Page Forward/Backward
Perpetual Calendar	Add/Delete Reminders
Continuous Reminders	Color or Monochrome
Five Reminders/Date	Twenty-five characters/Reminder
Saturday Sunday Highlighted	Runs Compiled or Interpretive

When you access the calendar, the dates with reminders appear in highlight. To call the reminders, simply move the cursor to the appropriate date. To enter reminders, just type away. Your program can be designed to call the calendar automatically if any reminders are due. Includes commented source code, operation instructions, and NO license or royalties. Optional TURBOSCREEN for instant screen display.

Demo Disk $10 **Source Code $675**

Also available: Generic Word Processor & Generic Calculator

Software Studios

8516 Sugarbush, Suite 104 • Annandale, VA 22003 • (703) 978-2339

Circle Number 24 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

However, it is the ability to lock and unlock records that excites me! The LOCK statement restricts access to a range of records within a "random" file. It has the following form:

```
LOCK [#]filenum [,[{record|[start] TO end}]
```

where filenum is the file whose records are being locked.

record is the number of the first record to be locked. LOCK with no arguments locks the entire file. If one record is specified, only that record is locked. If you specify a range of records and omit a starting record (start), then all records from the first record to the end of the range (end) are locked.

start TO end is an optional sequence designating the range of locked records.

The UNLOCK statement has the same format and is used to unlock a file before closing it. A matching record number range must be used in the UNLOCK statement: The two must match exactly.

Some examples:

LOCK #1	locks the entire file
UNLOCK #1	releases the entire file
LOCK #2,135	locks record number 135 in the #2 file
UNLOCK #2,135	releases that record and file
LOCK #3,TO 78	locks record numbers 1 thru 78 in file #3
UNLOCK #3,TO 78	releases the corresponding records and file
LOCK #4,55 TO 93	locks records 55 thru 93 in file #4
UNLOCK #4,55 TO 93	unlocks those records

On a sequential file, the lock and unlock statements affect the entire file, regardless of the range specified.

The way the LOCK/UNLOCK statement is defined by Microsoft, a program may be written that allows User-1 to access the same file as User-2, allowing each user to simultaneously update records within that file, provided they are working on different record numbers. If a user is working on record 49 of the file, when another user tries to lock that record, the error message "Permission Denied" is generated.

You can see how Microsoft has paved the way for programmers to use their compiled BASIC to write a powerful multi-user application. I've always liked BASIC for its ease-of-use and the fact that BASIC seems to be able to use every capability of a machine. (I've had great discussions along the lines of the challenge: "BASIC can do anything your language can do . . .") Perhaps, after examining the features of this compiler, you will want to return to BASIC to write your next application. □

Readers can reach Robert R. Shaffer at

Mind Matters, Inc.
31 North Sixth St.
Cottage Grove, Oregon 97424
503-942-1322

Generic Calculator

Developing a calculator module for your program? Why reinvent the wheel? Software Studios has developed a generic, full-function calculator written in BASICA with the following features:

Add/Subtract/Multiply/Divide	Fixed Decimal
Double Precision Operation	Printer Tape On/Off
Ten Digit Mantissa	Chain Calculations
On-screen Tape w/Total	Displayed Memory
Decimal Places 1 to 9	Clear/Clear Entry
Reverse Polish Notation	Add/Subtract Memory
Divide by Zero Trapped	Subtotal
Color or Monochrome	Runs Compiled or Interpretive
Overflow Indicator	

The calculator allows continuous, on-screen calculations, with 13-line display which clears on 14th entry with carry-over total. Easy, intuitive operation with on-screen commands. Optional TURBOSCREEN for instant screen display. Includes commented source code, operation instructions and NO license or royalties.

Demo Disk $10 **Source Code $625**

Also available: Generic Word Processor & Generic Calendar/Reminder

8516 Sugarbush, Suite 104 • Annandale, VA 22003 • (703) 978-2339

Circle 24 on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PROGRAMMING FOR PRINTER OUTPUT

review by Michael Hoyt

Programmers' Handbook of Computer Printer Commands

Editors Mary Lou & Fred B. East
Cardinal Point, Inc.
P. O. Box 596
Elletsville, IN 47429
812-876-7811
$37.95

Overview

This is a reference book for those who write programs that produce printed output. As the title clearly states, the target market for this book is programmers, although anyone thinking of buying a printer would find the book useful for making a feature-by-feature printer comparison.

The book has a pleasing "heft" because it's printed on heavier-than-normal paper and should hold up under heavy use. It is bound with a double-wire spiral and lies flat when open: a really nice feature for reference books. The book is not "glossy," but I don't think that detracts from it at all. In fact, I'm getting rather tired of getting only "packaging" when I buy a product, especially when I'm interested in "content." The *Programmers' Handbook of Computer Printer Commands* is "content," not "package." I like that.

Filling a Need

I think most would agree that the larger the market for a software package, the better its chances of being financially successful. If the market for your software can be expanded just by adding support for a greater variety of printers, it's likely you would pursue this relatively easy method of increasing your product's chances of survival.

I say "relatively easy" with good reason. If you have attempted software support of several different printer brands, you know that getting a manufacturer to part with an "evaluation unit" (loaner) or even a copy of a printer manual can be difficult. I've never quite understood why a manufacturer would not be willing to cough up a "loaner" for a few weeks so that his product could be supported by available software. Obviously, some are more worried that an "unauthorized party" might get a few days "free" use than they are about software support for their product.

The *Programmers' Handbook of Computer Printer Commands* is at least a partial solution to the problem. It can't offer hands-on testing but does make much-needed control code information readily available. With this information you can do preliminary programming. Then, as you have time, and can pry printers away from their manufacturers, do the "hands-on" testing. The book even helps a little here with its list of printer manufacturers (including addresses and phone numbers). The book's information about various printers makes it possible to structure printer support in an efficient manner. Rather than guess how the majority of printers are programmed, you can be certain with a little study of the book's tables. That makes it much simpler to design printer control modules, look-up tables, etc.

But even though the *Programmers' Handbook of Computer Printer Commands* is a useful addition to a programmer's library, it could be improved.

Table Format

My first complaint is with the structure of the code tables for each printer. This book is no different from most printer manuals in this respect. More times than not, printer control codes are listed in alphabetical order. This book's tables are no exception. Now, that's great for dictionaries because it makes it simpler to find specific words. But it's nonsense for control code tables.

If you know a specific control code in advance, fine. You can easily find it in an alphabetized table. But a programmer is usually looking for the code that enables a function. What is known in advance is the function, not the control code sequence. When a table is in alphabetical order, a programmer has to read the entire thing to find the correct code sequence.

It makes much more sense to list control codes grouped by function. It's then a simple matter of looking at one function group for the specific control sequence. Granted, printer manuals "traditionally" list codes alphabetically. But we're talking programmers here, not neophytes. In my opinion, if the editors expect this book to be accepted by programmers, then it should be formatted for programmers, not librarians.

Graphics

Graphics is the other area where the *Programmers' Handbook of Computer Printer Commands* could use some work. As micro computers get faster and more powerful, graphics capabilities are bound to be more and more in demand. I have no doubt that printed output of graphics will be one of the areas to grow very quickly.

Unfortunately, this book falls short when it comes to dealing with the graphics capabilities of printers. It's really too bad because by covering the area of graphics well, this book could quickly become indispensable to the programmer.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Missing is information concerning the maximum number of dot-columns printed per printer pass (line) and in at least one case, how to send graphics information to the printer (MSB high or low, 7- or 8-bit data, etc.). There are gaps in the "graphics" information which keep this book from being an all-inclusive control code reference for printer graphics.

If you already have a good understanding of the graphics capabilities of printers, then even with its graphics limitations, the *Programmers' Handbook of Computer Printer Commands* is useful as a ready quick-reference and memory refresher.

Printers Listed

Every available printer is not covered in this book. That is no fault of the editors. In fact, the number covered proves they made a truly herculean effort to get information from each manufacturer. The editors shouldn't be blamed for a lack of cooperation from manufacturers, rather they should be commended for putting up with the manufacturers' standard b.s.

It takes time to get a book to market, so of course the newest printers are not covered. I called one of the editors to see if new editions are planned and was happily surprised to find that the next is already underway. After they sent sample copies of the book to printer manufacturers, many of those originally "reluctant" are now being helpful. So the second volume will include information on the newest printers.

"Volume" is no mistake. Rather than do re-writes of the current edition, the editors intend to add new volumes, thereby building a "set" of printer reference books. Additional volumes will be added as needed to cover new printer models. As long as printers covered in this book are still being used, the *Programmers' Handbook of Computer Printer Commands* will retain its value.

Conclusion

Is the *Programmers' Handbook of Computer Printer Commands* what it claims to be? Yes and no. Yes, it is a compilation of printer control codes, and if you (as a programmer) are dealing with text modes only, then it is for you. But if you need information about the graphics capabilities of printers, then no, this book (current edition) leaves something to be desired.

Still, in spite of its problems, *Programmers' Handbook of Compter Printer Commands* is a useful reference book, and I'm adding it to my library. □

Model:

Control Code Table

Code	Hex	Decimal	Function
ESC I _CONT-@	5A 20 00	90 32 00	No Line Feed at buffer overflow (_=space character).
ESC D _CONT-@	44 20 00	68 32 00	Line Feed added at buffer overflow (_=space character).
ESC 6 CONT-@ 2	44 00 32	68 00 50	Ignores eighth bit of data byte.
ESC 7 CONT-@ 2	3A 00 32	70 00 50	Recognizes eighth bit of data byte.
ESC c	63	99	Restores standard instructions - (Software Reset).
ESC (a,b---n	28 a,b---n	40 a,b---n	Set Horizontal Tab line.
ESC u	75	117	Set Horizontal Tab in addition to those tabs already set (only one additional tab can be set in any one operation).
ESC) a,b---n	29 a,b---n	41 a,b---n	Clear Selected Horizontal Tabs.
ESC 0	30	48	Clear ALL Tabs.
ESC -	2D	45	Maximum width will be 8 dots.
ESC +	2B	43	Maximum width will be 16 dots.
ESC I	49	73	Start loading new character(s).
ESC '	27	39	Switch to custom character font.
ESC *	2A	42	Switch to custom character font (special-see below).
ESC $	24	36	Switch back to "normal" font.
ESC G nnnn	47 nnnn	71 nnnn	Print ix@ graphics columns corresponding to the following nnnn data bytes.
ESC S nnnn	53 nnnn	83 nnnn	Same as ESCAPE G.
ESC g nnn	67 nnn	103 nnn	Prints line corresponding to the following nnn x 8 data bytes.
ESC V nnnn c	56 nnnn c	86 nnnn c	Print nnnn repetitions of the dot column specified by c.
ESC F nnnn	46 nnnn	70 nnnn	Place succeeding printing nnnn dot positions from left margin.
CONT-H c the character c.	08 c	08 c	Backspace one character and print
CONT-N	0E	14	Begin Headline Mode.
CONT-O	0F	15	End Headline Mode.
CONT-_ n	1F n	31 n	Feed n lines of blank paper n=1,2,3,4,5,6,7,8,9,:,;,<,=,>,?
CONT-L	0C	12	Feeds paper to next Top-of-Form.

Write it once!
MasterFORTH
Portable programming environment

Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest. If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

MasterFORTH is a state-of-the art implementation of the Forth computer language. Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in macro assembler to make it even faster. MasterFORTH's relocatable utilities, transient definitions, and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint, and trace your way through most programming problems. A string package, file interface, and full screen editor are all standard features. And the optional target compiler lets you optimize your application for virtually any programming environment.

MasterFORTH exactly matches the Forth-83 Standard dialect described in *Mastering Forth* by Anderson and Tracy (Brady, 1984). The standard package includes the book and over 100 pages of supplementary documentation.

MasterFORTH standard package
- Macintosh $125
- IBM PC family (MS DOS 2.1)..... 125
- Apple II family (DOS 3.3) 125
- CP/M 2.x (IBM 3740 8")..... 125
- Commodore 64 (with graphics)... 100

Extensions
- Software Floating Point..... $60
- Hardware (8087) Floating Point ... 60
- Graphics (selected systems)..... 60
- Relocator (with utility sources)..... 60
- Target compiler (RAM and ROM) ... 350
- Target Application Generation System (TAGS) - MasterFORTH, Target compiler & relocator..... 495

Publications
- Printed source listings (each) $35
- Forth-83 International Standard 18

NEW PRODUCTS
MODEL List Processor..... $40
(with ELIZA and micro-LISP)

MICROMOTION
8726 S. Sepulveda Bl. #A1
Los Angeles, CA 90045
(213) 641-4710

Circle Number 13 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

ROMable CODE on PC!
PCLOCATE allows PC users to develop ROM-based software from MS-DOS "Exe" files. The user specifies the physical location of all segments. Output files are compatible with most PROM programmers. PCLOCATE supports the 8086, 8088, 80186, 80188, and 80286 processors. MC/VISA.

ALDIA SYSTEMS INCORPORATED
P.O. Box 37634
Phoenix, AZ 85069
(602) 866-1786

DATABASE EXPANDER
If you have TURBO TOOLBOX, expand it to 2,000,000,000 records! Your copy of TURBO TOOLBOX is automatically modified. You get a SUPER database plus all our 32-bit integer functions! We show you how to convert your existing database. $24.95 + $1 s&h.

Cauthorn & Associates
1811 N. Kaspar
Arlington Heights, IL 60004
(312) 259-3930

***PJ Billboard** is a product and services directory containing listings for hardware and software of interest to the IBM PC programmer.*

*Listings must be received typed, preferably on one of our convenient order forms. Copy deadlines are the same as the normal ad materials closing dates, which can be found on our rate card, or you may call the **Programmer's Journal** Advertising Department at 503-484-0602 for more information.*

SMARTWARE International, Inc., introduces:

- DISK MANAGEMENT FACILITY (DMF) including
- FILE SUPPORT FACILITY (FSF)
- Cosmetically appealing display of up to 24 files at a time;
- Single character entries to perform the following:

-Erase multiple files (up to 24 at a time);
-Unerase multiple files (up to 24 at a time) previously erased with this facility;
-Erase a complete directory of files and/or disk;
-Change file attributes;
-Type ASCII files to the screen;
-Select a directory for display;
-Browse files with update capability—view ASCII & HEX data (including hidden & system files);
-Dump viewed files to print w/range selection in an easy to read functional format;
-Run .EXE, .COM & .BAT files;
-Execute DOS commands;
-Rename, move or copy up to 14 files at a time, placing each in a different subdirectory or drive if desired.

- An extremely functional tool for users and technical personnel alike.
- A true value $49.95 (for both facilities) or priced separately at $39.95 each. MC/VISA

SMARTWARE International, Inc.
759 Avery Street
South Windsor, CT 06074
(203)646-2816

DESMET C $109
Full K&R C compiler for 8088/86/286 machines. Includes an assembler, linker, librarian, full screen editor (SEE[tm]), profiler, overlays and manual. Both 8087 and s/w floating point libraries included. PC/MSDOS-IBM format disks. **Rated 1st or 2nd** in all published benchmarks. Source level debugger $50 extra. DeSmet C for the Mac @ $150 includes machine-level debugger and SEE. VISA, MC, AMEX.

C Ware Corporation,
Dept. PJ
505 W. Olive, Suite 767
Sunnyvale, CA 94086
(408) 720-9696

MAKE $50
Uses both built-in and user supplied rules to specify how to construct your program(s). Compares directory dates and times to determine what files to update. Invokes compiler, assembler, linker or librarian to update files. Features: Install your own built-in rules, extensive macro capabilities, use DOS 2.x/3.x path names. VISA, MC, AMEX.

C Ware Corporation,
Dept. PJ
505 W. Olive, Suite 767
Sunnyvale, CA 94086
(408) 720-9696

BASIC PROGRAMMERS
Add **SALT & PEPPER** to your existing/new programs. Create dazzling Menus & Input screens like in dBASE & Lotus. Do Asc/desc. Sorts, PRT.SCR., Date processing, make your strings Roll and much more. **SALT & PEPPER** has 28 modules (in MS DOS compatible BASIC). They merge with your programs in minutes. **SALT & PEPPER** performs all the tricks, you get the applause. $59.95 ($2.50 s&h). Demo disk only $2.50 ppd MC/VISA.

COMPUTER GURU
40 Wagner Ave.
Piscataway, NJ 08854
(201) 356-6477

SEE[tm], A FULL SCREEN EDITOR $50. The same excellent editor we package with DeSmet C. Handles files larger than memory and can edit two files simultaneously. Its macro facility allows you to capture a series of keystrokes and replay them to ease repetitive tasks. Also featured: auto indent & brace/bracket/parenthesis matching to ease program entry. For the IBM. Note: For $59 more, you can have DeSmet C too! VISA, MC, AMEX.

C Ware Corporation,
Dept PJ
505 W. Olive, Suite 767
Sunnyvale, CA 94086
(408) 720-9696

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

AUTHOR GUIDELINES

We've always asked authors to submit their articles on disk with accompanying hard copy. Beyond that, we've had a long-standing policy that our authors shouldn't be burdened with a set of exacting criteria for article submission. Our feeling has been that it's up to us to see that a good article gets into our format and into print.

But because so many of you have asked us what format we prefer, we've come up with a short list that can serve as *PJ*'s author guidelines:

1. Submit the article on disk with an accompanying hard copy to

   Programmer's Journal
   P.O. Box 30160
   Eugene, OR 97403

2. Use the word processor of your choice. One that produces standard ASCII text files is easiest for us to work with (PC Write, Perfect Writer, Microsoft Word, for example). WordStar devotees, don't be discouraged by this preference of ours; we have a handy conversion utility furnished by Michael Hoyt that "cleans up" your files—that is, turns them into ASCII text files and deletes your formatting commands.

3. Don't indent paragraphs. Put your left margin in Column 1. Put two carriage returns between paragraphs.

4. Resist the temptation to use your word processor's formatting commands for boldface, underlining, etc. Indicate the emphases you want in red ink on the hard copy that accompanies your disk.

5. Use subheads to let the reader know what sections of your article are about. Put subheads flush left in upper and lower case.

6. Attach some information about yourself if you'd like it to appear at the end of your article. Let us know if you'd like us to print your address so that readers can get in touch with you.

Expect a response to your article submission within the month. Send us a stamped, self-addressed envelope if you'd like us to return your disk and manuscript.

Follow through: If you get some good letters in response to your article, forward them to *PJ*, and we'll print them in our LETTERS column.

ADVERTISER INDEX

	Page Number	Circle Number
Atron	8	1
Austin Code Works	42	30
Blaise	21	2
Dair Computer Systems	36	5
Data Base Decisions	6	6
Datalight	42	7
Ecosoft	Cov 4	8
Eigenware	42	29
Entelekon	3	9
Essential Software	16	10
Gimpel Software	20	11
Landmark Software	24	12
Martin Scot	10	27
Micromotion	51	13
Oliver Computing Company	35	15
Opt-Tech Data Processing	25	16
Peter Norton Computing	7	17
Phoenix Computer Products	4	18
Polytron	Cov 3	26
Quantum Information Systems	17	31
Raima Corporation	19	19
Rational Systems	44	20
Soft Advances	37	33
Softcraft	Cov 2	21
Software Horizons	46	23
Software Masters	34	32
Software Studios	48 & 49	24
Syscom Software	29	34
Turbopower Software	45	25
Zedcor	11	28

PUT IN A GOOD WORD

When you contact these Advertisers for information, please be sure to mention that you saw their product advertised in **Programmer's Journal**. Thanks!

I don't care if you ARE a programmer... after 24 years of marriage, it offends me to be referred to as your peripheral.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

VINTAGE dr

VOLUME I

- Vol. 1, No. 1: Norton batch files and Pascal I/O; Program style manual; COBOL on the IBM PC; Benchmarks; System analysis; BASIC *modules*; *Rounding error*.
- Vol. 1, No. 2: Defining character sets; C86 BIOS calls; Troubleshooting; Copy protection; Pascal code; Input routines; Writing an assembler.
- Vol. 1, No. 3: ASM dumb terminal; ASM tutorial routines; C source code; ASM graphics line drawing; Text editors reviewed; Utilities; Self-documenting programs; Text files to dBASE II; dBASE II PEEKS & POKES.
- Vol. 1, No. 4: Diskette directory via Pascal; MASM WHILE-WEND; C/ASM interface; ASM stay resident routine; Structured programming; Software law; Pascal tutorial; Default values and variable names.
- Vol. 1, No. 5: *Multi-tasking software review*; *Transporting* C programs; Programmer's calculator; dBASE II bargraphs; Utilities; ROM BIOS equates; Pascal check program; Combining modules in assembler; Structured programming in BASIC; Agreement outline.
- VOL. 1, No. 6: *ASM printer configuration*; DOS patch utility; Hi-speed graphics; Software ownership; Structured programming revisited; C and BASIC compared; Software protection; dWINDOW review.

VOLUME II

- Vol. 2, No. 1: Freelance consulting; Sorting in C; Pascal text editor source code; CRT distress; Compiled BASIC; Assembly modules for C; Designing a relational database; Ultra Utilities; Punctuation and Style reviewed.
- Vol. 2, No. 2: Interview with Brian Kernighan; Understanding C through assembly language; *Utility routines in C* and assembly; Music in C; Automatic line number optimizing in BASIC; Assembly language tutorial; Full-screen input routines in BASIC; dBASE II printer control codes; Pascal tools reviewed; C programming support products.
- Vol. 2, No. 3: *Software development environments*; Creating a productive screen handling tool; Assembly tutorial; Generating graphics within windows; Composition and editing system for musical ideas; Hybridizing assembly and BASIC; Creating screens; Linking Assembler to Pascal and BASIC.
- Vol. 2, No. 4: Writing a custom ROM BIOS for PC DOS; RISC, transputers, and mass storage; The future of disk protection; 8088 assembly macros; Random access files in BASIC; C utility routines you can buy; using the PMATE macro language; Choosing a database manager; E/SP syntax-checking; Pascal editor reviewed.
- Vol. 2, No. 5: Developing the APX CORE Executive; From Assembler to C via a 211 byte .EXE main( ); Scroll control using BASIC and Assembler; Faster screen handling with BASIC and ASM, Random access files (Part two); RTOOL, a PC-resident programmers toolkit; dBkey reviewed; Pascal's pointer power.
- Vol. 2, No. 6: Periscope—building a hardware board for the IBM PC; Intel 80286 bugs by FNE; AT&T surprises all; Field Usage and pointer subtraction in C; Calendar functions for BASIC; Memory segmentation; Reduce your .EXE file sizes; Enhancing Debug; dBASE II to dBASE III? THINKTANK reviewed.

VOLUME III

- Vol. 3, No. 1: November's Comdex; C—its uses and abuses; Using the C preprocessor; Memory segmentation examples; Index files (Part 1); Structured programming language; Periscope, a symbolic debugger.
- Vol. 3, No. 2: UNIX! (again); Artificial Intelligence; Doctor C's tips—pointers to functions; Calendar date routines in BASIC, Calculate record layouts, automatically!
- Vol. 3, No. 3: NAPLPS and the professional programmer; Instant-C; Printer control from a batch file; Repositioning the stars in Starfinder; Reducing EXE file size through data positioning; tools for better programming.
- Vol 3, No. 4: More UNIX stuff; C programming books (Part One); C code excerpts (Part One); How to cure the 'dreaded locked keyboard' condition; Calling MS DOS from Turbo Pascal; DOS 2.X sort and find filters; the SET Tools.
- Vol 3, No. 5: Windows done here; C programming books (Part Two); C code excerpts (Part Two); A menu-driven PC using *DISPLAY.COM*;Turbo Pascal and MASM: making them work together; dBASE II routines; Turbo Pascal 3.0; TopView Programmer's Toolkit.
- Vol. 3, No. 6: About C's HLL overhead; How to recover all your costs; Report on the COMDEX and Usenix shows; The Santa Cruz Operation XENIX for the IBM PC; Common C bugs; CTOOLS2; Turbo Pascal 3.0 and subdirectories; More dBASE II routines; Clipper reviewed; A better way to interface BASIC and Assembler.

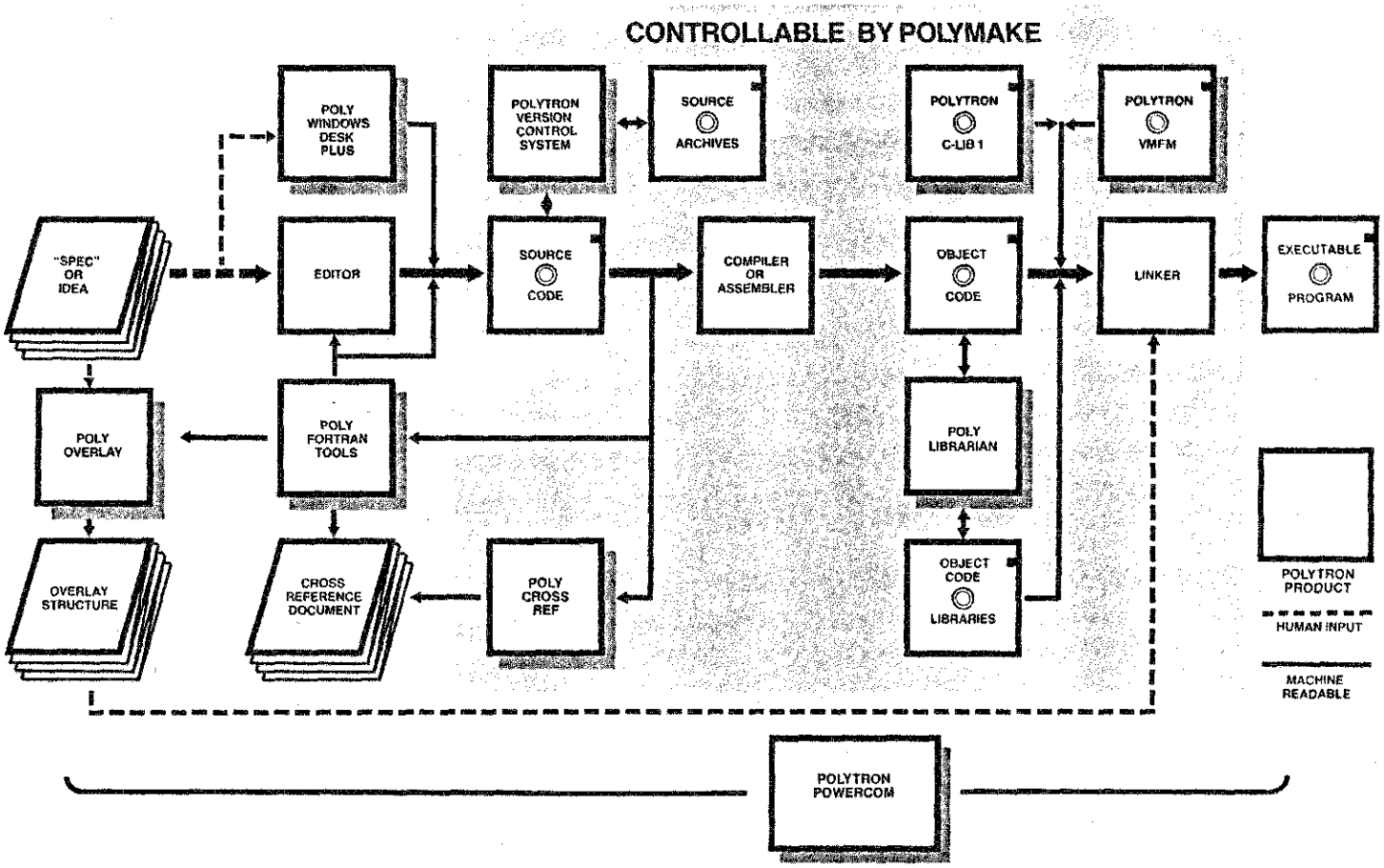
VOLUME IV

- Vol. 4, No. 1: PC/AT hard disk problems and plunging 64K DRAM prices; AUTOINC explained; Sorting using an AVL tree; Combining binary trees; Using DOS to call programs from Turbo Pascal; An operating system for the people; Final installment of Brett Salter's dTOOLS; Two microcomputer graphics books reviewed.

To order back issues, please use the form on the next page.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

POLYTRON TOOLS IN THE SOFTWARE DEVELOPMENT PROCESS



Thousands of Professional Software Developers with demanding deadlines turn to high performance MS-DOS/PC-DOS tools from POLYTRON to boost their productivity. All the tools shown above are explained in detail in the **POLYTRON Programmer's Catalog.**

Network & Site Licenses Available.

PolyWindows DeskPlus
The Modular, Expandable Desktop Organizer
100% memory-resident desktop organizer preferred by programmers. Configurable so you can have one or more of the following functions available instantly: Hex/Decimal/Binary Calculator, Standard Calculator, ASCII Table, Multiple Document Text Editor, PolyKey for creating your own keyboard macros, Rolodex-type files, Calendar, Alarm Clock, Appointment Books, Autodialer, Grabber for cut-and-paste between applications. **Copy-Protected $49.95, Not Copy-Protected $84.95.**

POLYTRON PowerCom
The PC Communications/Terminal Emulation Program for Programmers
This is the program that is getting rave reviews from communications experts. Programmers can now use the most powerful and flexible script language available in the PC communications industry. Features YMODEM, XMODEM, Kermit, Advanced Kermit, X.PC and B Protocols. Provides VT100/ANSI, VT52 and ADM3a terminal emulation. The features list is lengthy so we recommend the $15.00 Demo Disk and Evaluation Packet. **The New Price/Performance Leader at only $179.**

POLYTRON Version Control System (PVCS)
Efficiently maintains revision history of source files for software project management. Maintains chronological, historical records of changes as "edit scripts" (reverse deltas) with full text of the latest "checked in" revision. Reconstructs any prior "revision" of any module, defines a "version" as specified revisions of various modules, supports branching from prior revisions, optional password protection. Friendly, unobtrusive user interface is based on a unique "syntax guide" concept with context-sensitive help. Designed for single or multiple programmer projects. A powerful tool with mainframe power. **Single-user license: $395.**

PolyCrossRef
The Multi-Lingual Cross Reference System
Generates Cross Reference listings across single or multiple source files of one or more languages. Tells you exactly where each variable and procedure is defined and everywhere it is referenced. **Package includes PolyXREF engine plus one source language module plus .CRF module: $129, additional source language modules [C, Pascal, Assembler or English]: $49 each. Complete package including all four language modules: $219.**

PolyLibrarian
The Object Module Library Manager
The premier library manager allows you to create, examine and manage libraries. Modules can be listed, added, replaced, extracted, and deleted. You can search for public or external names and even change them. Three operating modes can be intermixed for maximum flexibility. **PolyLibrarian I for Microsoft format libraries: $99, PolyLibrarian II for Microsoft and Intel format libraries: $149.**

PolyMake
The Intelligent Program Builder and Maintenance Tool
Automates the software maintenance process. Determines which files (programs, etc.) are out of date and invokes your compiler, linker, librarian and does whatever is necessary to bring your entire system up to date. Frees you from the need to remember which files depend on others and which files have been modified. Remembers the exact sequence of operations necessary to create a new revision. Completely automatic unlike competing products. Once you use it, you can't live without it. **Only $99.**

POLYTRON C Library 1
Over 65 High Performance Routines For Lattice C Compiler Users
These are routines for serious programmers that need Executive and I/O functions. Complete source code (mostly Assembler) included. **Only $99.**

POLYTRON Virtual Memory File Manager (PVMFM)
Provides efficient virtual file access and buffering for both text and binary files with automatic buffer-to-disk swapping. Reduces program access time and handles data structures larger than memory allows. PVMFM is a library of user-callable functions that may be linked with your applications code to provide virtual file management capability and random access file buffering. Compatible with Assembler and high-level languages. **Only $199.**

To Order Products Call
1-800-547-4000
Ask for Dept. No. 306
Foreign & Oregon orders call
(503) 684-3000
Send Checks, P.O.s To:
POLYTRON Corporation
P.O. Box 787 DS-306
Hillsboro, OR 97123
Add $5.00 Shipping To Total Order

To Order detailed Programmer's Catalog, send requests to address above for fast response (or circle reader service card number).

POLYTRON

High Quality Software Since 1982

Circle Number **26** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NEW RELEASE

Ecosoft's Eco-C88 Rel. 3.0 C Compiler

$59.95

Release 3.0 has new features at an unbelievably low price. ECO-C88 now has:

- Prototyping (the new type-checking enhancement)
- enum and void data types
- structure passing and assignment
- All operators and data types (except bit fields)
- A standard library with more than 200 functions (many of which are System V compatible for greater code portability)
- cc and mini-make that all but automates the compile process
- 8087 support (we sense the 8087 at runtime – no dual libraries)
- ASM or OBJ output for use with MSDOS linker
- Tiered error messages – enable-disable lint-like error checking
- Fast compiles and executing code
- Expanded user's manual
- Enhanced CED program editor (limited time offer)

We also offer the following support products for Eco-C88.

CED Program Editor — $29.95

CED now supports on-line function help. If you've forgotten how to use a standard library function, just type in the name of the function and CED gives you a brief summary, including function arguments. CED is a full screen editor with auto-flagging of source code errors, multiple windows, macros, and is fully configurable to suit your needs. You can edit, compile, link, and execute DOS commands from within the editor. Perfect for use with Eco-C88. For IBM PC, AT and look alikes.

C Source for Standard Library — $10

($20 if not with order)

Contains all of the source code for the library functions that are distributed with Eco-C88, excluding the transcendentals and functions written in assembler.

Developer's Library — $25

($50 if not with order)

Contains the source code for all library functions, including the transcendentals and those written in assembler. Perfect for the developer that wish to write their own custom functions or learn how we implemented the Eco-C88 library.

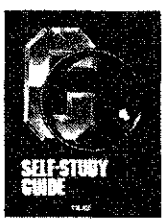
ISAM Library — $15

($30 if not with order)

Contains the code from the **C Programmer's Library** in relocatable format (i.e., .OBJ) including the delete code for the ISAM file handler.

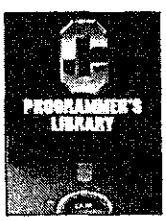
C Programming Guide — $20

After reading the 1st edition, Jerry Pournelle (**BYTE** Magazine) said: "I recommend this book . . . Read it *before* trying to tackle Kernighan and Ritchie." The second editon expands this best seller and walks you through the C language in an easy-to-understand manner. Many of the error messages include references to this book making it a perfect companion to Eco-C88 for those just starting out with C.



C Self-Study Guide — $17

(Purdum, Que Corp.). Designed for those learning C on their own. The book is filled with questions-answers designed to illustrate many of the tips, traps, and techniques of the C language. Although written to complement the Guide, it may be used with any introductory text on C.



C Programmer's Library — $20

(Purdum, Leslie, Stegemoller, Que Corp.). This best seller is an intermediate text designed to teach you how to write library functions in a generalized fashion. The book covers many advanced C topics and contains many useful additions to your library including a complete ISAM file handler.

Eco-C88 C compiler requires an IBM PC, XT, or AT (or compatible) with 256K of memory, 2 disk drives and MSDOS 2.1 or later. Call today:

1-800-952-0472 (for orders)
or
1-317-255-6476 (tech. info.)

ECOSOFT INC.

Ecosoft, Inc.
6413 N. College Ave. • Indianapolis, IN 46220

AMERICAN EXPRESS · MasterCard · VISA

TRADEMARKS: ECO-C88, ECOSOFT

Circle Number 8 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED